Computer Simulation Experiments in Phonetics and Phonology

Simulation Technology in Linguistic Research on Human Speech

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik
der Universität Stuttgart zur Erlangung der Würde eines
Doktors der Philosophie (Dr. phil.) genehmigte Abhandlung

Vorgelegt von
Daniel Duran
aus Ostfildern

Hauptberichter:	Prof. Dr. phil. habil. Grzegorz Dogil
1. Mitberichter:	Prof. Dr. Hinrich Schütze, Ph. D.
2. Mitberichter:	Prof. Dr. phil. Bernd Möbius

Tag der mündlichen Prüfung: 21. Juni 2013

Institut für Maschinelle Sprachverarbeitung
2013

Diese Arbeit ist meiner Omi MARICA TRUPELJAK (* 1936 – † 2009) gewidmet,
die mich schon als Kind zur Informatik an der Universität Stuttgart
mitgenommen hat, wo ich nun promovieren durfte.

Acknowledgments

First of all, I would like to thank Professor Grzegorz Dogil for giving me the chance to write this doctoral dissertation and for his inspiration and guidance throughout this journey.

This thesis is based in part on work from the collaborative research centre SFB 732, project A2, funded by *Deutsche Forschungsgemeinschaft* (DFG). In particular, chapters 4, 5 and 6 build on that work. I would like to thank Professor Hinrich Schütze and Professor Bernd Möbius for letting me join the A2-team, for many inspiring discussions and for giving me the freedom to work on this thesis. Without the collaborative research centre the research presented in this thesis would not have been possible. I am grateful to all my colleagues in the SFB and at the *Institute for Natural Language Processing* (IMS) — thanks for the fun and inspiring environment! Special thanks to Jagoda Bruni, Michael Walsh, Antje Schweitzer and Natalie Lewandowski for always having time to discuss scientific matters and more.

Thanks to Prof. Dr. Martine Grice and Dr. Doris Mücke for facilitating the EMA recordings at the Institute of Linguistics at the University of Cologne. I have used that Polish corpus in several of my simulation experiments (cf. appendix A.1). Thanks to Prof. dr hab. inż. Grażyna Demenko for the permission to use the Polish BOSS corpus in my experiments (cf. appendix A.3).

Thanks to Dr. Thomas Kemp at Sony Deutschland and all past and present members of his entire Speech and Sound Group for continued support, inspiration and a great working environment. My work at Sony provided not only an additional perspective to my dissertation, it also allowed me to deepen my understanding on many fascinating topics and ideas—some of which are reflected herein.

Finally, I would like to thank all of my friends and family, especially my brother Mihael Duran and my parents Ruža and Ivan Duran for everything!

Contents

List of Figures

List of Tables

List of algorithms

Listings

Abbreviations, symbols and notational conventions

Acronyms

ANN artificial neural network (see section 3.3 on page 68)

ASCII American Standard Code for Information Interchange

ASR automatic speech recognition

CSM Context Sequence Model (see section 6.2 on page 220)

CSV comma-separated values (file format)

CV consonant–vowel

CCV consonant–consonant–vowel

EMA electromagnetic midsagittal articulography

HMM hidden-Markov-model

ICPhS International Congress of Phonetic Sciences

IPA International Phonetic Alphabet (also: International Phonetic Association, see following section)

MFCC mel-frequency cepstral coefficient

NLP natural language processing

SOM self-organising map (see section 3.4.1 on page 70)

V vowel

VC vowel–consonant

VCC vowel–consonant–consonant

Notation

The following conventions and symbols are used in this thesis.

Transcriptions

The symbols of the *International Phonetic Alphabet* are used for all phonetic and phonemic transcriptions according to the handbook of the IPA (the International Phonetic Association, 1999). I follow Jassem (2003, p.103f) for the transcription of Polish — except for the open mid vowels, for which I prefer the symbols "ɛ" and "ɔ", respectively[1], in both phonetic and phonemic transcriptions. Phonetic symbols are used without enclosing brackets if they appear in tables and figures, otherwise transcriptions are marked as follows:

Notation	*Meaning*
/…/	Enclosing oblique lines indicate a (broad) phonetic, phonological or phonemic transcription.
[…]	Enclosing square brackets indicate a (narrow) phonetic transcription.
⟨…⟩	Enclosing angle brackets indicate an orthographic transcription.

Mathematical notation and other Symbols and Abbreviations

The notational conventions are in part adopted from: Manning et al. (2009).

Notation	*Meaning*
✓	yes; true; applies (in tables)
✗	no; false; does not apply (in tables)
$\lvert A \rvert$	Cardinality of a set A (the number of elements in set A)
A	Sets are denoted by italic upper-case Roman letters

continued on next page...

[1] This usage is consistent with the phonetic openness of these vowels and the standard IPA vowel chart (cf. the International Phonetic Association, 1999, p.12). It avoids potential confusion originating from the usage of different symbols for phonetic [ɛ] and [ɔ] and corresponding phonemic /e/ and /o/ for the same vowels. Note also that Polish is not illustrated in the IPA handbook.

Notation	*Meaning*
cm	centimetre = 10^{-2} m
cf.	confer
e.g.	exempli gratia — for example
et al.	et alii — and others
etc.	et cetera
F_1, F_2, F_3	First, second and third formant
Hz	hertz
i.e.	id est — that is
iff	if and only if
kHz	kilohertz = 10^3 Hz
m	metre
ms	millisecond = 10^{-3} s
$\mathbb{N}$	The set of natural numbers
$p(A\|B)$	Conditional probability of A given B; for sequences this denotes the probability of item A preceded by item B
p.	page *(in references)*
p.7f; p.7ff	pages 7 and the following; pages 7 and the following pages
$\mathbb{R}$	The set of real numbers
s	second
$\vec{x}$	A vector is denoted by an italic lower case Roman letter with an arrow accent.

Source code listings and in-line examples of program code or names of programs, functions or methods are set in a type-writer font, e.g. `experiment1`.

Zusammenfassung

Die gesprochene Sprache ist ein kontinuierliches, höchst variables und ambiges akustisches Signal welches wir als Abfolge von Lauten, Silben und Wörtern wahrnehmen. Im Erstspracherwerb stehen Kinder vor der Aufgabe bedeutungstragende sprachliche Einheiten aus dem kontinuierlichen Redefluß zu extrahieren. Die Phonetik und die Phonologie befassen sich mit den kleinsten sprachlichen Einheiten, ihrer Produktion im menschlichen Sprechapparat, ihren Eigenschaften als akustisches Phänomen und ihrer Perzeption sowie mit den in einer Sprache verwendeten Spracheinheiten und ihren möglichen Kombinationen. In der experimentellen Phonetik oder der Laborphonologie greift man Dabei auf Korpora größerer Mengen von Sprachaufnahmen zurück oder untersucht spezielle Phänomene gezielt an Versuchspersonen im Labor.

Die vorliegende Arbeit befasst sich mit Computersimulationsexperimenten, einer dritten Herangehensweise in der Phonetik und Phonologie. Der Untertitel der Arbeit betont den weiteren Kontext der Simulationstechnologie in der sprachwissenschaftlichen Forschung zur menschlichen gesprochenen Sprache. Die *Natürlichkeit* der Daten ist vergleichsweise niedriger als bei korpusbasierten Untersuchungen oder Laborexperimenten. Dafür bieten Computersimulationen dem Experimentator den höhsten Grad an Kontrolle über die Daten und das Experiment. Dieser spezielle Ansatz Fragen der Phonetik und Phonologie mit Hilfe von Simulationstechnologie zu untersuchen wird in der Forschergemeinde nur von einer Minderheit verfolgt. Es gibt bisher noch keine umfassende Arbeit die sich mit dieser Methodik befasst, einen systematischen Überblick zum aktuellen Stand der Forschung bietet sowie die Möglichkeiten in der Phonetik und Phonologie diskutiert.

Kapitel 1 bietet eine Einführung in das Thema dieser Dissertation. Ein Ziel dieser Arbeit ist es einen repräsentativen Überblick über die Verwendung von Computersimulationsexperimenten in der Phonetik und Phonologie zu bieten. Darüber hinaus werden neben der Linguistik und Computerlinguistik zahlreiche Forschungsdisziplinen als für diese Arbeit relevant betrachtet, welche sich mit der menschlichen Sprachperzeption und -produktion befassen. Dazu gehören insbesondere die mathematische Psychologie und die Kognitionswissenschaften. Wichtige Methoden und Forschungsansätze für die Simulationstechnologie in der Erforschung der menschlichen Sprache sind auch in der Maschinellen Sprachverarbeitung, der Sprachtechnologie und Sprachsignalverarbeitung zu finden sowie in Teilbereichen der Informatik wie der Künstlichen Intelligenz und dem maschinellen Lernen. Aufgrund dieser interdisziplinären Breite des Themas beinhaltet diese Arbeit einen umfangreichen Literaturüberblick repräsentativer Publikationen.

Dass der Einsatz von Simulationstechnologie in der Erforschung der Sprache keineswegs neu ist, zeigt der kurze, historische Überblick am Anfang von Kaptiel 2. Frühe Vorläufer lassen sich in Konstruktionen zur künstlichen Erzeugung von Sprachlauten finden – zunächst mit mechanischen und später elektronischen Mitteln. Mit der Entwicklung der ersten digitalen Rechner wurde dann die Funktion solcher speziellen Geräte mit Hilfe von Computerprogrammen nachgebildet. Die Arbeit von Liljencrants and Lindblom (1972) kann als eigentlicher Beginn des hier behandelten Forschungszweigs der Phonetik und Phonologie angesehen werden in welchem Computersimulationen eingesetzt werden.

Welche Forschungsfragen mit Hilfe von Simulationsexperimenten untersucht wurden wird in einem umfangreichen, thematischen Literaturüberblick in Kapitel 2 dargestellt. Zunächst werden Arbeiten betrachtet, die sich mit Lautsystemen und ihrer Entstehung und Evolution befassen. Eine bedeutende Forschungsfrage stellt dabei auch der Erwerb von Lautsystemen dar. Dazu gehören sowohl der Erwerb des muttersprachlichen Sprachlautsystem wie auch der Erwerb zweit- und fremdsprachlicher Systeme. Hierbei können Simulationsexperimente dazu eingesetzt werden, bestehende theoretische Modelle und Hypothesen zu testen und gegen empirische Daten zu vergleichen. Computersimulationen können verwendet werden um Lautsysteme als Ganzes zu Untersuchen oder auch spezifischere Modelle zu phonetischen und phonologischen Strukturen, wie etwa der Silbenstruktur. Verschiedene Modelle zur Sprachperzeption und -produktion werden vorgestellt. Modelle zur Perzpetions-Produktions-Schleife und der Entwicklung des Sprechens lassen sich mit Computersimulationsexperimenten untersuchen welche künstliche neuronale Netze und eine artikulatorische Sprachsynthese kombinieren. Computersimulationsexperimente erlauben so Untersuchungen von empirisch schwer zu fassenden oder völlig unzugänglichen Phänomenen. Als weitere Beispiele werden Arbeiten zum Sprachwandel und zur historischen Linguistik diskutiert. Die Entwicklung eines sprachlichen Systems über mehrere Generationen hinweg lässt sich empirisch nur schwer erfassen. Der Ursprung der menschlichen Sprache erscheint experimentell vollkommen unzugänglich. Mit Computersimulationen lassen sich die anatomischen wie kognitiven individuellen Voraussetzungen für die Produktion und Perzeption gesprochener Sprache anhand verschiedener Modelle untersuchen. Außerdem lassen sich gesellschaftliche Faktoren in der sprachlichen Interaktion modellieren und in Simulationsexperimenten testen und über beliebig viele Generationen hinweg simulieren. Der Überblick zu bestehenden Arbeiten zeigt einerseits wie unterschiedliche Forschungsfragen mit denselben Methoden behandelt werden können. Andererseits werden dieselben Forschungsfragen oft mit ganz unterschiedlichen Methoden untersucht. Ein Überblick zu häufig eingesetzten Methoden und grundlegenden Konzepten wird daher gesondert in Kapitel 3 gegeben. Hierzu gehören vor allem Methoden und Grundlagen des maschinellen Lernens, künstliche neuronale Netze und Prinzipien der Selbstorganisation.

Der zweite Teil dieser Dissertation stellt eine Reihe meiner eigenen Computersimu-

lationsexperimente vor und diskutiert relevante Arbeiten für die jeweils untersuchten Phänomene.

Kapitel 4 präsentiert Experimente zum Thema der Sprachsegmentierung und der Frage, wie der Mensch diese Fähigkeit anhand der zur Verfügung stehenden Informationen erlernen oder erwerben kann. Ein zentrales Problem in diesem Zusammenhang ist das der *Grenzhaftigkeit* ("*boundariness*"). Im Rahmen dieser Arbeit wir dieser Begriff für die kontinuierliche Natur segmentaler Information an jeder Stelle im Redefluß verwendet. Eine Reihe von Experimenten wird präsentiert. Diese beschäftigen sich mit der Frage, wie die Fähigkeit der Sprachsegmentierung anhand des kontinuierlichen Sprachsignals ohne Rückgriff auf externe Informationen wie der Kenntnis des Lautinventars der gegebenen Sprache erworben werden kann. Relevante Arbeiten aus diesem Bereich werden diskutiert, die vor allem aus dem Bereich der Psycholinguistik und Kognitionswissenschaften kommen. Als wichtiges Problem im Bereich der Sprachsegmentierungsmodelle wird insbesondere die Evaluation von entsprechenden Computersimulationen hervorgehoben. Dazu wird eine Untersuchung von Evaluationsmaßen für die automatische Sprachsegmentierung vorgestellt. Verschiedene objektive Qualitätsmaße werden verglichen und anhand verschiedener Kriterien auf ihre Eigenschaften hin untersucht. Neben gängigen Qualitätsmaßen wie dem *F-Maß* oder der *Übersegmentierungsrate* stelle ich dabei auch zwei neue Maße vor und untersuche zusätzlich noch zwei bisher nur in der Textsegmentierung eingesetzte Maße (P_k und *WindowDiff*).

Kapitel 5 befasst sich mit der Frage, wie sprachliche Einheiten zu Kategorien und Gruppen von ähnlichen Einheiten zusammengefasst werden können, wenn sie erst einmal aus dem segmentierten Sprachstrom extrahiert wurden. Methodisch basieren die vorgestellten Experimente auf der Clusteranalyse. Soweit dies im Rahmen dieser Dissertation notwendig ist, werden die Grundlagen des *Clusterings* vorgestellt und relevante Arbeiten besprochen. Schwerpunkt der Experimente in Kapitel 5 ist die Untersuchung kontinuierlicher akustischer Sprachaufnahmen im Vergleich zu entsprechenden kontinuierlichen artikulographischen Aufnahmen. In einem ersten Experiment wird die im Signal enthaltene linguistische Information unterschiedlicher Segmentklassen untersucht. Vokale und Konsonanten zeigen dabei unterschiedliche Resultate für akustische und artikulatorische Daten. Im zweiten Experiment werden Vokale in unterschiedlichen Silbenkontexten untersucht. Dabei zeigt sich, dass sich kontextuelle Informationen über Silbengrenzen hinweg nachweisen lassen.

Kapitel 6 stellt eine Reihe von Experimenten vor, die sich mit der Frage befassen, wie das "*Context Sequence Model*" um eine artikulatorische Dimension bzw. Modalität erweitert werden kann. Relevante Arbeiten zur Exemplartheorie und Sprachperzeptions- und Produktionsmodellen werden diskutiert. Die Experimente verfolgen einen neuen Ansatz indem artikulographische Aufnahmen in einem Produktionsmodell in derselben Art verarbeitet werden wie akustische Sprachaufnahmen. Die Simulationsergebnisse zeigen nicht nur, dass eine derartige Verwendung von artikulographischen Daten möglich ist. Es

zeigt sich auch, dass die Qualität der Produktionen ausgehend von artikulographischen Daten vergleichbar ist mit Produktionen, die auf akustischen Daten basieren. Teilweise liefern artikulographische Daten auch bessere Ergebnisse. Nimmt man an, dass Sprachproduktion und Sprachperzeption gekoppelt sind, so muss man in einem vollständigen Modell die Perzpetions-Produktions-Schleife berücksichtigen. Interne Repräsentationen von sprachlichen Einheiten oder die in der Sprachproduktion verwendeten Informationen lassen sich empirisch nicht direkt untersuchen. Wie in Kapitel 6 gezeigt lässt sich in Computersimulationsexperimenten der Einfluss verschiedener Arten von Repräsentationen direkt mit einander vergleichen und isoliert untersuchen.

Insgesamt bietet diese Dissertation einen umfangreichen Überblick über eine vielversprechende Herangehensweise in der Erforschung der gesprochenen Sprache. Die vorliegende Arbeit ist damit die erste dieser Art in der Phonetik und Phonologie. Mit Computersimulationsexperimenten lassen sich nicht nur aufwendige empirische Studien komplementieren und teilweise auch ersetzen. Die Simulationstechnologie bietet auch die Möglichkeit theoretische Modelle direkt zu untersuchen und die Interaktionen ihrer Bestandteile zu studieren. Somit lassen sich bestehende Hypothesen testen und neue generieren.

Abstract

Speech is a continuous, highly variable and ambiguous acoustic signal which we perceive as a sequence of sounds, syllables and words. Infants face the task in first language acquisition to extract meaningful linguistic units from the continuous speech stream. Phonetics and phonology are concerned with the smallest of these units, with their production in the human articulatory apparatus, with their properties as acoustic phenomena and with their perception as well as with the speech units used in a given language and their possible combinations. In experimental phonetics and laboratory phonology corpora with large amounts of speech recordings are used. Alternatively, certain phenomena are studied specifically with human subjects in a laboratory setting.

This present thesis is concerned with computer simulation experiments — a third kind of approach in phonetic and phonology. The subtitle emphasises the broader context of simulation technology in linguistic research on human speech. The data are less *natural* in comparison to corpus based studies or laboratory experiments. In return, computer simulation experiments offer the highest degree of control over the data and the experiment to the experimenter. This specific approach to questions in phonetics and phonology using simulation technology is taken only by a minority of the research community. There is no comprehensive work which addresses this methodology and offers a systematic overview of the state of the art and discusses its possibilities in phonetics and phonology.

Chapter 1 provides an introduction to the topic of this dissertation. One goal of this thesis is to give a representative overview of computer simulation experiments in phonetics and phonology. Moreover, in addition to linguistics and computational linguistics, a number of research disciplines are identified as being relevant for the subject of this thesis which are concerned with human speech perception and production. Among these are in particular computational psychology and cognitive sciences. Important methods and research approaches using simulation technology for the study of human speech can also be found in natural language processing and speech signal processing as well as in areas of computer science such as artificial intelligence and machine learning. Due to this interdisciplinary breadth of the topic, this thesis comprises a comprehensive overview of representative publications.

The application of simulation technology in research on human speech is not new. This can be seen from the short historical overview which is offered at the beginning of chapter 2. Early precursors can be found in constructions for the artificial creation of speech sounds — first by mechanical means and then electronically. With the devel-

opment of the first digital computers, the functionality of such specialised machines was reproduced with computer programmes. The work by Liljencrants and Lindblom (1972) can be regarded as the actual beginning of the branch of research in phonetics and phonology that is investigated here which makes use of computer simulations.

Chapter 2 provides a detailed thematic overview of the literature and presents the various research questions which have been studied with computer simulation experiments. First, works are addressed which focus on sound systems, their emergence and evolution. An important research question in this context is the acquisition of sound systems. This includes the acquisition of the native sound system as well as the acquisition of any second of foreign language system. Simulation experiments can be employed in this area to test existing models and hypotheses and compare them against empirical evidence. Computer simulations can be used to study sound systems as a whole or for the study of more specific models of phonetic and phonological structures, like the syllable structure. Various models of speech perception and production are presented. Models of the speech perception–production loop and the speech development can be studied with computer simulations which combine artificial neural networks with articulatory speech synthesis. Computer simulation experiments thus allow the investigation of phenomena which are empirically hard to capture or even totally inaccessible. As additional examples in this line, works on language change and historical linguistics are discussed. The development of a language system over many generations is only hard to capture empirically. The origin of human language seems completely inaccessible to empirical inquiry. Individual anatomical and cognitive requirements for the perception and production of speech can be studied with computer simulations on different models. In addition, social factors in linguistic interactions can be modelled and tested in simulations over arbitrary numbers of generations. The overview of existing work shows on the one hand how different research questions can be addressed by the same methods. On the other hand, it shows how the same research questions can be addressed by a variety of different methods. An overview of often applied methods and basic concepts is therefore given separately in chapter 3. These are especially methods and foundations of machine learning, artificial neural networks and principles of self-organisation.

The second part of this thesis presents a series of my own computer simulation experiments and discusses relevant works for the addressed phenomena.

Chapter 4 presents experiments on speech segmentation and addresses the question how humans can achieve this faculty based on the available information. A central problem in this context is that of *boundariness*. In the scope of this thesis, this term is used to denote the continuous nature of segmental information at any position in the speech stream. A series of experiments is presented. These address the question of how the speech segmentation faculty can be acquired based on the continuous speech stream without reference to external information like the speech sound inventory. Relevant work in this area is discussed which comes primarily from the fields of psycholinguistics

and cognitive sciences. An important problem of speech segmentation models which is emphasised is the evaluation of computer simulations. Different objective quality measures are compared and their properties are investigated according to various criteria. Apart from conventional quality measures like the *F-measure* or the *over-segmentation rate*, I propose two new measures and additionally investigate two quality measures which so far have been applied to text segmentation (P_k and *WindowDiff*).

Chapter 5 addresses the question of how speech units can be grouped into categories once they are extracted from the segmented speech stream. Methodologically, the presented experiments are based on cluster analysis. The foundations of *clustering* are presented as far as possible within the scope of this thesis and relevant work is discussed. The focus of the experiments in chapter 5 is the investigation of continuous acoustic speech recordings in comparison to corresponding continuous articulographic recordings. In the first experiment, the linguistic information contained in different segment classes is investigated. Vowels and consonants show different results for acoustic and articulatory data. Vowels in different syllabic contexts are investigated in the second experiment. The experiment shows that contextual information can be found across syllable boundaries.

Chapter 6 presents a series of experiments which address the question of how the *Context Sequence Model* can be expanded by adding an articulatory dimension or modality. Relevant work on exemplar theory and speech perception and production models are discussed. The experiments take on a new approach in which a production model treats articulatory recordings in the same way as continuous acoustic speech recordings. The results show not only that continuous articulographic data can be processed in such a way. They also show that the quality of the productions based on articulatory data is comparable to the quality of productions based on acoustic data. The results may even be improved by articulatory data, to some extent. Assuming that speech perception and production are coupled, a complete model has to take the perception–production loop into account. Internal representations of speech units or information employed during speech production cannot be investigated directly in empirical studies. As shown in chapter 6, the effects of different kinds of representations can be compared directly in computer simulation experiments.

This dissertation offers a comprehensive overview of a promising approach to the study of spoken language. The present thesis is as such the first of this kind in phonetics and phonology. Computer simulation experiments can not only complement laborious empirical studies and replace them to some extent. Simulation technology also offers the potential to directly study theoretical models and the interactions of their components. This allows for testing of existing hypotheses and the generation of new ones.

1. Introduction

> "...and the time may well come when computers will be as familiar to linguists as they are now to physicists and astronomers."
>
> (Lamb, 1961, p.412)

This dissertation provides a detailed investigation of simulation experiments in phonetics and phonology which complement both empirical and theoretical studies. The subject of study of this thesis covers various disciplines. On the one hand there are the various sub-disciplines and fields of research in phonetics and phonology and, additionally, the related fields of research on human speech perception and production from adjacent disciplines like psychology, cognitive science, psycho-linguistics and other linguistic disciplines. On the other hand, there are the various fields of research in computer science like machine learning, artificial intelligence or speech technology.

One goal of the research presented in this thesis is to provide a representative overview of the use of computer simulation experiments in phonetics and phonology. An attempt was made to review the various publications representative of the simulation methodology. Some of the researchers cited in this thesis make extensive use of computer simulation experiments, while others have only a few publications on this issue.

I will not treat phonetics and phonology as strictly separate research disciplines in this thesis. Moreover, I consider both as proper branches of linguistics. Accordingly, the computer simulation studies discussed in this thesis could be considered as applications of computational linguistics.

This introductory chapter is structured as follows: Section 1.1 elaborates on the motivation for this dissertation. Section 1.2 defines some essential terminology and section 1.3 further narrows down the scope of this present work.

1.1. Motivation

After more than half a century of research on artificial intelligence and machine learning and a similarly long history of research on speech technology and natural language processing (NLP), there is a huge repository of knowledge, methodological paradigms and tools from which phoneticians and phonologists can choose for their design of

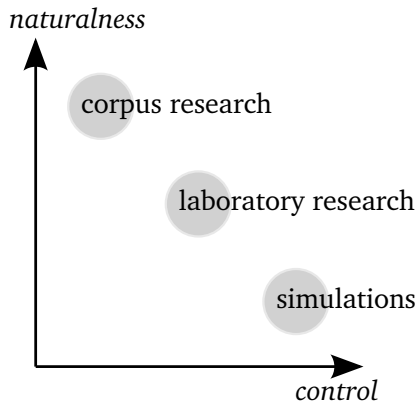


Figure 1.1.: Allocation of computer simulation experiments relative to corpus-based and laboratory experiments with respect to data naturalness and degree of experimental control.

computer simulation experiments. Fant (2004, p.B-24) reviews the development of phonetics and phonology in the second half of the 20th century. He notes that "in the second period, 1965 to 1980, computers had become a standard laboratory tool, which marked the transition from analogue to digital processing in all aspects of speech research, [...]".

Valuable insight can also be gained from speech and language related theoretical and methodological considerations in the field of computational psychology. A substantial part of this thesis relates to work from cognitive sciences and computational psychology and related disciplines. Related work will be discussed in this thesis depending on the respective research subject without a priori limitations to publications labelled as phonetics or phonology.

Figure 1.1 illustrates a possible two-dimensional classification scheme of research methods in phonetics and phonology. The two given dimensions are *naturalness* of the data and the degree of *control* of the researcher over it. It is common practice in experimental phonetics to record speech samples of some test subjects in a laboratory setting. This offers a high degree of control over the data. On the other hand, it can be argued that the naturalness of speech data recorded in a laboratory environment is not given, especially if intrusive measurements are involved. Another common practice is the phonetic analysis of speech corpora, i.e. large data sets of recorded and usually annotated speech recordings. Such speech corpora can provide a higher degree of naturalness, while the control of the investigator over the data is much more restricted. Within the two-dimensional scheme of figure 1.1, these two approaches can be located relative to each other such that corpus research is higher in data naturalness and laboratory research is higher in data control.

A third kind of approach to phonetics and phonology can be placed in this figure such

that it is higher in control and lower in naturalness than the other two methods. This third type is the subject of this thesis: simulation technology. This specific approach to phonetics and phonology is currently practised only by a minority of the research community. At the 14th International Congress of Phonetic Sciences (ICPhS), held in San Francisco in 1999, there were at least 16 computer simulation related papers — out of a total of 642, i.e. approx. 2.5%. At the 17th ICPhS, held in Hong Kong in 2011, at least 12 papers presented work which used some kind of computer simulation experiment — out of a total of 579 papers, i.e. approx. 2%. Only two papers related to phonetics using articulatory speech synthesis were presented at Eurospeech 1997. At Interspeech 2011 there were approximately 8 papers (not counting purely speech/signal processing and speech technology oriented work like ASR or text-to-speech synthesis).

To the best of my knowledge, there is no comprehensive discussion of the topic of simulation technology in phonetics and phonology[1]. It is necessary to reach beyond the limits of the fields of phonetics and phonology in order to get a better overview of the importance of computer simulation experiments for the linguistic study of speech and language. Therefore, an important aspect of this present thesis is the consideration of related theoretical and methodological issues as they have been discussed in neighbouring scientific fields such as computer science or computational psychology. In some disciplines, there is a wealth of work on theoretical and methodological aspects of computer simulations which is worth taking into account in a global review of this particular research method.

1.1.1. Potential advantages of computer simulation studies

To paraphrase Henke (1966): An unimplemented model is empty of purpose[2]. He stresses the necessity to explicitly and consistently define the model when it comes to computer simulations (p.15). Thus, he argues, actually implementing a model as a computer simulation, helps avoiding "hand waving" explanations (p.14). The implementation of computational simulations forces the experimenter to explicitly spell out implicit assumptions of the underlying model or theory. Sun (2008a, p.6) writes in his introduction to computational cognitive modelling: "Computational models provide algorithmic specificity [...]. Therefore they provide both conceptual clarity and precision."

Simulation experiments are not only useful tools for investigations of what is predicted by a given model or theory, they are also particularly useful for discovering what is not

[1] I dare to say that, if there is such a published work, it is not widely known and it did not have a great impact in the fields of phonetics and phonology.

[2] "The test of any model is how well does it explain and/or describe that process which it proclaims to model. Thus it must be 'implemented' so that its behaviour or 'outputs' for given inputs can be observed and compared with the real system. An unimplemented model cannot be so tested and thus seems quite empty of purpose." (Henke, 1966, p.14)

predicted. Baker (2008, p.289) notes: "Computational studies frequently bring to light hidden implications of theories [...]". He discusses the apparent paradox that models are expected to broaden knowledge while they are only encoding the knowledge that the modeller already has. The purpose of a simulation experiment, thus, is broadening the knowledge about the relationships and interactions of the variables of a system and its model. Similarly, one might claim that computational simulations of speech production/perception or any other phonetic or phonological processes and phenomena cannot tell us anything about language or the human speech production and perception faculty. It has been argued, for example, that the human brain is essentially an analogue computer which cannot be described or modelled by a digital computer (cf. Kohonen, 2001, p.76). On the other hand, simulations can provide valuable insights into the implemented models and the tested hypotheses. They can visualise unexpected or previously unknown consequences of the underlying theoretical assumptions on which the implementation of a simulation is based. The aim of scientific work in general can be described as designing a model of a system which is assumed to be the cause of an observed phenomenon and validating the model such that its resulting consequences must agree with the observed phenomenon.

Simulations can be used to identify unnecessary assumptions or parameters in models. Sometimes, some phenomenon or postulated property of a model can be shown to emerge automatically from other model parameters. By applying *Occam's razor*, the model with the fewer necessary assumptions should be preferred. Boersma and Hamann (2008), for example, used simulation experiments within the framework of Optimality Theory to show that "dispersion" in sound systems arises automatically after a number of generations without the need for an explicit rule to control it (see section 2.2).

The basic idea of carrying out experiments is to *isolate* the subject of interest as much as possible in order to be able to investigate its details. The problem with experimental phonetics or laboratory phonology is that it is not possible to isolate specific parts of the linguistic system from the speakers. As Planck emphasised that an experimental setup poses a specific question to nature[3], experiments with computer simulations can pose a variety of questions to unobservable or inseparable aspects of phonetics and phonology and, in particular, to the tested model itself. With the aid of computer simulations, the researcher has the opportunity to devise experiments, which address abstract theoretical aspects of phonetics and phonology, independent of any particular language or speaker.

Wedel (2004b, p.10) points out the usefulness of computer simulations when it comes to *self-organising systems* (see section 3.4 on page 70). It can be close to impossible to predict the behaviour of such systems even in cases where total knowledge about their constituting elements and the starting conditions is given.

[3] "...jede Versuchsanordnung stellt die spezielle Formulierung einer gewissen Frage an die Natur dar." (Planck, 1931, p.25)

1.2. Terminology

This thesis investigates computer simulation experiments in phonetics and phonology. But what exactly is meant by the term "simulation"? I will discuss some terminology in this section and establish the context for the following chapters. Note, that a complete formal treatment of the terms discussed in this section is far beyond the scope of this thesis. Some terms will necessarily be left unspecified and just used according to common practice in the literature.

The term *simulation* is used in this thesis to refer to a research technique or method of analysing a given system based on an algorithmically implemented model of that system. A *system* in that sense is any formal or real-world object with an internal structure or a set of connected, interacting elements. Simulations are often used to study dynamic systems and their behaviour which cannot be (easily) described in formal or mathematical terms.

A *model* is informally defined by Manning and Schütze (1999, p.55) as a theoretical construct used to "explain something in the world". Kohonen (2001, p.72) defines the term *model* as a finite set of variables and their quantitative interactions. He defines the term *method* accordingly as something effectively working in applications which may or may not be derived from a model. Following the glossary provided by Kohonen (2001, p.396), the term *simulation* can be defined alternatively as imitation of the behaviour of a system by its model.

It is possible to distinguish various types of simulations. One distinction that can be made is that between *static* and *dynamic* simulations. A static simulation is a simulation of a time-invariant system, while time is an important factor in dynamic systems. Another distinction is that between *deterministic* and *stochastic* simulations. The former are simulations of deterministic systems, i.e. systems which are not subject to random events. A stochastic simulation, on the other hand, incorporates some element of randomness. I will treat the term *probabilistic simulation* as a synonym for the term stochastic simulation. A *Monte Carlo simulation* is considered here as a special type of stochastic simulations (cf. section 3.5.1). Computer simulations often use pseudo-random numbers which can be efficiently generated algorithmically and also re-generated in the same order, if necessary. The latter fact is important for the exact repetition of an experiment and the reproduction of its results. This renders such experiments to be actually deterministic in a strict formal sense. However, I will adopt the common terminology and refer to simulations which are *supposed to* incorporate random values as stochastic simulations.

Dynamic simulations can be time continuous or time discrete. In the latter case, they can be termed *event-based* or *round-based* dynamic simulations since they simulate the states of the system at discrete time intervals depending on events associated with the respective time interval.

A *multi-agent simulation* is a simulation of a set of individual, independent interacting agents. An *agent* in this context is an independent object with an internal state and

certain methods to receive input and produce output.

With respect to the type of data which the simulations operate on, they can be classified as either *continuous valued* or *discrete valued* simulations. I will refer to the latter type as *symbolic simulations*. These are, for example, simulations on a set of items with associated, discrete features or labels. Note that the distinction between discrete and continuous data is not always clear. Consider, for example, the following questions: Is the representation of angles by one degree intervals discrete or continuous? Is a digitised speech recording with its fixed sampling rate and fixed range of possible values discrete or continuous? Mathematically, these are of course discrete data representations.

Finally, the term *computer simulation* refers to a simulation implemented such that it can be executed as a programme on a computer given a parametrisation of the underlying model. Other terms, which I will treat as synonyms, are: *computer model*, *computational model*, or *numerical experiment* (and other possible permutations).

1.3. What is not in the scope of this thesis

The body of research literature discussed in this thesis covers a wide range of different disciplines. The term *simulation* can have quite different interpretations, as can be seen in the previous section. This section further narrows down the subject of this thesis by definition of what is *not* in its scope.

A borderline case are *computer aided* experiments such as perception experiments with manipulated speech data. The *purpose* of an experiment is probably more important in such cases than its methodology. Computer simulation experiments with synthesised speech may be evaluated by humans listening to the results. However, the same experimental setup can also be used to investigate the participants' perception. In this latter case, I would not speak of a simulation experiment according to the definitions given above.

Simulations need not be (1) implemented as computer programs or (2) use computers. Speech synthesis, for example, can be achieved by mechanical or electrical means using specialised hardware — the so called *speaking machines* (cf. section 2.1.1). An example of simulation experiments which are not based on computers are laboratory experiments designed for the elicitation of (pseudo-) spontaneous speech. Lewandowski (2012, p.126), for example, reports on a laboratory study which aimed at creating conditions "as close to a natural scenario as possible". Such *social simulations* are not considered in this thesis.

At the time the first digital computers became available, the term "simulation" was often used to refer to the new possibilities of replicating the functionality of electronic experimental equipment by means of a computer programme (cf. section 2.1 for a historical overview). David et al. (1959), for example, discuss "simulations" as the implemented functionality of (analogue) signal processing equipment on a digital

computer for the acoustic analysis or processing of speech signals:

> “Simulation of equipment with digital computers results in reduction of the time and cost of experimentation, and more important, achieves a gross expansion of the ideas which can be considered.”
> (David et al., 1959, p.113)

While the simulation of speech processing hardware is not in the scope of this thesis, the second point mentioned also applies to computer simulation experiments in the fields of phonetics and phonology: Using computer simulations allows for investigations which would be hard or even impossible to carry out otherwise.

Such kinds of computer simulations of hardware devices will not be considered in this thesis. Using digital recordings of speech and computers to record, store, analyse or synthesise speech is the standard approach in experimental phonetics or laboratory phonology. Moreover, calling the utilisation of a computer for the analysis of a speech signal a “simulation” would seem strange to today’s researchers.

The title of this thesis identifies the subject of this thesis as computer simulation experiments in phonetics and phonology. According to the definitions given above, the simple term *simulation* could be used. However, to avoid terminological confusion, I use the expressions *computer simulation experiment* or *computer simulation study* throughout the text of this thesis.

1.4. Outline

This thesis consists of two main parts: first, a comprehensive, general overview and second, a presentation of my own experiments. The remainder of this thesis is structured as follows:

Chapter 2 gives a first thematic overview of related work which applies computer simulation experiments to study spoken language. Chapter 3 gives a second overview focusing on methodological issues. My own experiments are then presented in the following three chapters. Related work on the specific topics addressed by my own experiments is also discussed in the second part of this thesis. Chapter 4 presents related work on speech segmentation and a series of my own simulation experiments. First, evaluation measures are examined in detail. Then an unsupervised approach to the acquisition of speech segmentation is presented. Chapter 5 presents related work on data clustering and presents a series of computer simulation experiments on unsupervised classification of speech items. Chapter 6 presents computer simulation experiments which extend the Context Sequence Model (CSM). Chapter 7 discusses the work presented in this thesis and concludes the main part. Appendix A gives an overview of the speech databases used in my experiments. Finally, appendix B lists the commented source code of my experiments presented in this thesis.

2. Computer simulations in phonetics and phonology

This chapter gives a thematic overview of computer simulation experiments in phonetics and phonology, the historical development and the state of the art. This chapter also includes related work from other disciplines where simulation technology has been applied to study human speech, its properties, perception and production.

Section 2.1 gives a brief historical overview, describing the beginnings of computer simulations and their application in linguistic research and speech and language technology. Section 2.2 reviews some simulation studies on sound systems. Section 2.3 reviews work on phonetic on phonological structure. Speech perception and production models are discussed in section 2.4. Simulation studies on language change are reviewed in section 2.5. Finally, section 2.6 addresses the topic of articulation and speech synthesis.

Some of the works discussed here had a great impact in phonetics and phonology, while others might not be so well known to a broader audience.

Speech segmentation is discussed in detail in chapter 4. Section 4.1 gives an overview of related work on speech segmentation in general. The problem of assessing the quality of a generated segmentation is discussed in section 4.2. Exemplar-theoretic models of speech production and perception are discussed in detail in chapter 6. These topics are not considered separately in this chapter.

2.1. History

This section gives a brief historical overview of the beginnings of computer simulations in various research areas and the close relations between the progress in hardware development, computer science and phonetics and phonology. The primary focus is, as throughout this thesis, on computer simulation experiments in studies on speech and language.

2.1.1. Speaking machines

The beginnings of what today is called *speech synthesis* can be traced back to myths and anecdotal reports about speaking statues. The earliest documented examples of the artificial creation of speech sounds is usually associated with the *speaking machines* of

Kratzenstein or von Kempelen (Ohala, 2011; Trouvain and Brackhane, 2011). The idea of speaking objects seems to be as old as recorded history itself. Attempts at creating such objects were made by constructing mechanical machines. The constructors of such machines tried to simulate the process of articulation and incorporated the available anatomical knowledge of their time. In an article on the history of mechanical simulation of speech, Fagyal (2001, p.293) puts forth the hypothesis that "designs of speaking machines similar to the ones exhibited in the 18th century did in fact exist earlier". Fagyal (2001, p.297) analyses the 1667 edition of *La Science Universelle*, a textbook on modern science by Charles Sorel, sieur de Souvigny. There, a hypothetical "speaking machine" is described as a musical instrument with pipes and a keyboard. The analogy of speech production with aerophone "makes sense from an articulatory point of view", as Fagyal (2001, p.297) notes: both produce sound by a (constricted) stream of air.

> "...17th century inventors' knowledge about the mechanical reproduction of sounds had to come from musical instruments, because grammar books could not be of much use in this respect."
> (Fagyal, 2001, p.297)

One of the sources underlying this description of a speaking machine is traced to Marin Mersenne and his work *Harmonie Universelle, contenant la théorie et la pratique de la Musique* (1636) who also wrote on speaking machines and reported on constructing an instrument to produce vowels (Fagyal, 2001, p.303ff). The sources of Sorel's "description of a semi-automatic, multilingual speaking machine modeled after a hydraulic organ, with the potential of using unlimited vocabulary" are traced to 17th century writings on science and mechanics. This suggests that the first attempts at constructing speaking machines were in fact made one century earlier than the documented construction of the first speaking machines.

In 1780, Christian Gottlieb Kratzenstein won a competition of the St. Petersburg Academy of Sciences with a speaking machine that could mechanically produce five vowel sounds. In 1781 he published his monograph on vowel synthesis in which he also describes the anatomy of the vocal tract (Ohala, 2011, p.157). Trouvain and Brackhane (2011, p.164) point out that von Kempelen's 1791 book *Mechanismus der menschlichen Sprache* has "great historical relevance for the phonetic sciences". They write that the experience of a mechanical instrument which produces human-like sounds "inevitably leads to the core of the phonetic sciences: 'How is it that humans are able to speak?'" (p.165). The important difference between von Kempelen's speaking machine and that of Kratzenstein is the fact that von Kempelen tried to imitate the human speech production apparatus.

2.1.2. Transitions

A huge progress in speech synthesis was facilitated by analogue electronic devices. Already at the first ICPhS, Trautwein (1932) discussed methods of electronic synthesis of speech sounds (and music). He notes:

> “Er [Hermann] nimmt an, dass die Vokale durch Stosserregung von gedämpften Schwingungen, den sogenannten Formanten in der Mund- und Nasenhöhle gebildet werden. Wenn man die Hermannsche Theorie mit elektrischen Schwingungen nachbildet, kommt man wie der Versuch bestätigt zu einer sehr guten Vokalimitation.”
> (Trautwein, 1932, p.200)

This is an early example of how a simulation (although not computer based) is used to test and verify a given model. Implementing a given formant model of vowel timbre, Trautwein reports that the result is a “very good vowel imitation”. Meyer-Eppler (1949) presents a comprehensive overview of electrical and electronic speech and sound synthesis devices. He refers to work by Toulon which could be considered one of the first examples of concatenative speech synthesis (p.118) and he already addresses the non-invariance problem (p.119; see also section 2.4). At the fourth ICPhS (Sovijärvi and Aalto, 1961), speech synthesis by analogue electronic means was still the state of the art (cf. Cooper, 1961; see also section 2.6).

Simulations of hardware devices are explicitly excluded from the subject of this thesis (see section 1.3). They represent the beginning of a transition “from analogue to digital processing” as Fant (2004, p.B-24) put it. With the availability of the first digital computers for academic research, linguists began exploring the new tool and discussing its potential. The term “simulation” was often used in this context to refer to the new possibilities of replicating the functionality of electronic equipment by means of a computer programme. The term “simulation” is used in this sense, for example, by David et al. (1959). They discuss the implementation of functionality of analogue signal processing equipment on a digital computer for the acoustic analysis or processing of speech signals. They point out that this approach not only reduces the cost and time needed to prepare speech experiments, but that it also allows for new ideas to be considered which would have been hard or even impossible to implement otherwise. Schroeder (1969, p.1078f) discusses the use of a “new tool” in an “old science”: namely, the computer and its applications in acoustics. The author reviews progress in the area of speech analysis through the use of digital computers. “Simulations” are mentioned but the term has a different meaning which corresponds to today’s use of the terms *experimental phonetics* or *speech processing*.

2.1.3. Computers, simulations and mechanolinguistics

As soon as digital computers were available they have been used for simulation experiments. Enrico Fermi had the first ideas of Monte Carlo simulations in the 1930s (cf. section 3.5.1). The first computer simulation experiments were carried out in 1947 by Stanisław Ulam and John von Neumann on the ENIAC, one of the first digital computers. Interestingly, these first computer simulation experiments can be considered simulations of a hardware device: they basically replicated the functionality of the MONTE CARLO TROLLEY, or FERMIAC. This was an analogue computer constructed to do Monte Carlo calculations in nuclear physics (Metropolis, 1987, p.129).

The idea to imitate human-like behaviour on a machine by replicating human developmental processes (i.e. language acquisition and speech perception and production in the case of this thesis) is not new. Turing (1950) proposed the famous *imitation game* — often referred to as *the Turing test* — to test whether it would be possible to build a machine which can verbally perform as well as a human player in the same position of the game. The game is about three players, where the interrogator cannot see the other two players who are supposed to be male and female, respectively. The interrogator knows that one of the other players lies and has to decide which of the two is male and which female, based only on the players' answers to questions of any kind[1]. The test for human-like intelligence of a computer is done by replacing one of the two human players by a computer. The computer would then pass the test, if its performance in the game would be indistinguishable from that of a human player. Turing speculates on whether and how such an intelligent machine could be built and poses the question:

> "Instead of trying to produce a programme to simulate the adult mind, why not rather try to produce one which simulates the child's?"
> (Turing, 1950, p. 456)

Turing's question seems to be reflected by recent research on human language and speech acquisition using computational models and simulations. Much of this work comes from other scientific disciplines than phonetics or phonology, e.g. psycho- or neurolinguistics or branches of computer science. As far as it is related to human speech production and perception, I will include such work in the discussions of this thesis.

The earliest applications of simulation technology to a problem related to phonetics and phonology is the artificial production of spoken utterances, i.e. speech synthesis (cf. section 2.6). One of the first speech synthesizers implemented on a (IBM 7090) computer was presented by Kelly and Gerstman (1961). One part of their "talking machine" is described as a simulation of "a more or less conventional resonance synthesizer[...]". They explicitly point out that "the scheme is simple enough to permit realization with analog hardware".

[1] Note, that Turing excluded speech from the game, arguing that "[i]n order that tones of voice may not help the interrogator the answers should be written, or better still, typewritten" (Turing, 1950, p.434).

Lamb (1961) suggests using computers as linguistic research tools. After introducing the new tool, its structure and usage in some detail (pp.383–409), he discusses possible applications in "*mechanolinguistics*" — the (at that time) new "interdisciplinary field which links linguistics and computer science".[2] Among the uses of computers in linguistics, he lists two areas of particular relevance to the theme of this thesis. He refers to the first one as "linguistic automation" and to the second as "simulation of language dynamics", and describes the former as follows[3]:

> "This area involves having the computer *use language* to a limited extent; that is, it is concerned with the simulation of language-using behavior."
> (Lamb, 1961, p.410)

On simulation of language dynamics he writes:

> "The primary type of study in this area would involve placing models of specific languages in the machine and applying to them certain proposed general principles of linguistic change or certain kinds of external influences, to see what the resulting language would actually look like."
> (Lamb, 1961, p.411)

Interestingly, it seems that Lamb did not believe in the possibility of automatic phonetic transcription of a speech signal (i.e. automatic speech recognition (ASR)). He writes: "Except for such things as listening to speech and recording it in a phonetic transcription, there is no reason why anything in the area of linguistic analysis should a priori be considered immune to treatment by mechanical means" (p.409f).

An early example of the use of computers in "statistical phonology" is the work at Brown University mentioned by Kučera (1962, p.279f). As examples of possible applications of computers (in what today could be called computational linguistics), Kučera points out:

> "The programming library of this project now includes programs capable of performing a mechanical phonemic transcription of Russian and of Czech (from the conventional orthographic representation), syllabic analysis, various frequency counts, and the determination of statistical distributions of phonemes and phoneme groups."
> (Kučera, 1962, p.281)

Some possibilities in the field of speech synthesis are discussed by Ladefoged (1964, p.205ff). He speculates about the advantages of constructing a hardware articulatory speech synthesizer. The envisioned hardware synthesizers would have been simpler in some respects as compared to computer simulations because, for example, they would not have required detailed mathematical knowledge about sound propagation and the acoustic consequences of all possible vocal tract shapes. At the same time, he

[2] Lamb (1961, p.409) notes that an alternative name for that scientific discipline could be "computational linguistics". As he was concerned about the connotation of "numerical calculations", he favoured the term "mechanolinguistics".

[3] Emphasis from the original text.

argues, they would have been more useful for the investigation of articulation, as they would have required a more precise knowledge of the forces that are responsible for all possible articulatory gestures. Ladefoged (1964, p.213) emphasises that the HASKINS PATTERN PLAYBACK[4] synthesizer was at that time the most widely used device and that "it has produced the most experimental results because of its conceptual and operational simplicity"; and that there is a "great need for a speech synthesizer which uses an articulatory description of speech in an equally simple way". Computer simulations on an articulatory model were presented by Henke (1966), although this was not yet a speech synthesis system[5].

Garvin (1967, p.174) describes "computer experiments" in which the computer simulates the part of the human analyst, eliciting information from a human informant. Today, these envisioned interactive online questionnaires are part of standard methodology. In this view, the computer simulations which are in the focus of this thesis could be considered accordingly as *simulations of informants*.

The beginning of the modern age of computer simulations in phonetics and phonology is associated with the seminal work by Liljencrants and Lindblom (1972) on vowel systems (see section 2.2). They investigated the factors influencing vowel systems by running computer simulations which incorporate principles of self-organisation. The resulting structures predicted by the simulations are compared against empirical data from existing vowels systems in various natural languages. Liljencrants and Lindblom discuss the relation between phonetics and phonology (and linguistics in general) and the divergence of the two. Phonetics, as they argue, is primarily concerned with the structure of speech and the interpretation of linguistic form. Based on their simulation experiments and the finding that phonological structure can emerge from low level interactions, they propose to integrate phonetics and linguistics:

> "Suppose, however, that we let the horse and cart change places. Rather than accept it as a-priori, we attempt to derive linguistic form as a consequence of various substance-based principles pertaining to the use of spoken language and its biological, sociological, and communicative aspects."
> (Liljencrants and Lindblom, 1972, p.859)

2.2. Sound Systems – Emergence, competition and change of linguistic categories

One of the first, and probably the most prominent, example of vowel system structures being investigated by means of computer simulations is the seminal work by Liljencrants

[4] http://www.haskins.yale.edu/featured/patplay.html (accessed 2012-12)

[5] An articulatory synthesizer as a research tool for computer simulation studies on speech articulation was presented by Mermelstein (1969). This system was reportedly capable of producing intelligible speech.

and Lindblom (1972). They examine the emergent structures of vowel systems consisting of 3 to 11 vowels. Vowels in their model are represented by points in a two-dimensional, delimited space. Each simulation starts with the specified number of vowels placed at equal distance from the centre of the space. The programme then iteratively seeks to maximize the mutual contrasts between the vowels. This is achieved by defining a "total energy measure" as:

$$E = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} \frac{1}{r_{ij}^2} \quad , \tag{2.1}$$

where $r_{ij} = (x_i - x_j)^2 + (y_i - y_j)^2$ is the distance between two vowel points i and j, represented by their two coordinates, denoted x and y, respectively (p.842). The vowels are moved around inside the vowel space by forces of repulsion until a minimum for E is found. Liljencrants and Lindblom (1972, p.859) argue for an approach to linguistics which is "substance-based" considering primarily spoken language and all its aspects. Their simulation experiment is understood as one example implementing that proposal.

Computer simulations of category maintenance and change often consist of a pair of speaker–hearer agents which repeatedly exchange some abstract speech items (e.g. Wedel, 2004a; Boersma and Hamann, 2008). These agents either take turns in producing and categorising perceived speech items based on their internal lexicon which is constantly updated. Or, they represent an adult and a child agent, which interact such that the child agent acquires its interpretation of the system of the adult. Some simulations are also based on a single speaker–hearer agent who feeds back its own output to itself in a perception–production loop (e.g. Wedel, 2004b, p.23ff; see also section 2.4).

Wedel (2004a, p.1) presents computer simulations of category competition in an exemplar-theoretic framework. He argues that *contrast* is not "a property of forms" but that it can be described rather in an exemplar-theoretic framework as being implicitly driven by the statistical association of forms to categories. Wedel puts forward the hypothesis that "contrast maintenance is driven through category competition between form-meaning pairings". Given this hypothesis he then addresses the question of why distinctive features seem to be the minimal units of contrast in phonology and not morphemes (which are the smallest meaning-bearing units, i.e. units with a "form-meaning" association).

Boersma and Hamann (2008) present a series of computer simulation experiments within the framework of Stochastic Optimality Theory. They show that dispersion effects in sound systems emerge automatically after a number of generations without the need for an explicit rule. Specifically, the simulation uses only "cue constraints" and "articulatory constraints" — which are presupposed to be necessary independent of dispersion effects (p.226). Computer simulations of the acquisition of the English /s/–/ʃ/ system, represented by their spectral means (p.232), is described as well as a

simulation of the Polish chain shift from /ʃ sʲ s/ to /s̻ ɕ s̺/ (p.250). In the simulations, the learner starts with a set of cue constraints with initially equal ranks. During an acquisition phase, it adapts its internal constraint ranks according to the perceived input using the *Gradual Learning Algorithm*. In multi-generation simulations dispersion effects emerge after a few generations of repeated acquisition periods. In these simulations, a learner goes through an acquisition period after which it produces input for the second generation. Note, that agents represent not individual speaker-listeners but entire generations (p.246). Boersma and Hamann discuss in detail the basic assumptions on which their model is based and contrast them with other approaches.

Goldsmith and Xanthos (2009) focus on phonotactics and use *unsupervised learning* methods to investigate how the distinction between the two sound categories *vowel* and *consonant* can be learned from distributional information of phoneme symbols taken from a corpus. They compare three different computational approaches. The first one is an algorithm by Sukhotin, who was the first to propose a language-independent algorithm to distinguish vowels and consonants in a symbolic representation (cf. Guy, 1991). The algorithm is based on occurrence and co-occurrence frequencies and the assumption that vowels are most frequent and that they generally co-occur more often with consonants than with other vowels. Without explicitly advocating any specific phonological theory, Goldsmith and Xanthos focus on the most fundamental concepts: the distinction of phonemes into vowels and consonants and, after having "discovered" them (p.23), on vowel harmony and syllable structure.

2.2.1. Acquisition of speech sounds

Probably the "first unsupervised acquisition of phonetic structure" by a machine is presented by Coen (2006, p.1451). He uses an unsupervised clustering method to learn American English vowel categories from raw acoustic and visual data (i.e. from formant values and lip contour data). No assumptions about the number of vowel categories or their distributions are made (p.1456). This approach is motivated by considering the fact that the infant does not receive *labelled data* but has to learn about categories and their distributions based on raw, perceptual input. This work suggests that it is indeed possible to acquire a vowel system (at least the American English vowel system) based only on formant patterns and simultaneous lip contours.

A connectionist model of speech sound acquisition is presented by Kello and Plaut (2004) (cf. section 3.3). An important aspect of this data-driven model is that it does not impose any segmental structure on the input (p.2356). Kello and Plaut (2004, p.2356) argue that if the incorporation of information about the segmental structure of the input is crucial for the ability of a model to learn phonological representations, such a model does not "scale to handle all of the complexities inherent in the development and processing of real speech".

A computational model which aims at explaining two phenomena of rapid adaptation of phonetic categories — phonetic recalibration and selective adaptation — is presented by Kleinschmidt and Jaeger (2011b). The first effect which they simulate is "perceptual learning" or "perceptual recalibration". This is the phenomenon of adapting existing categories such that an initially ambiguous item becomes associated with one particular category. The second effect is selective adaptation where a phonetic category is narrowed due to repeated presentation of an unambiguous item. Kleinschmidt and Jaeger use *Bayesian belief updating* to model these two perceptual phonetic phenomena (see section 3.2). They compare two different types of models. The first one, the "unimodal model", is based on acoustic cues, and the other one, the "multimodal model" (p.12), is based on phonetic cues which incorporate audio-visual information. The unimodal model is a Gaussian mixture model of the phonetic categories. In the multimodal model, phonetic cues are represented by a weighted sum of the auditory and the visual cues for a given percept (p.16). The comparison of their simulations with data elicited from empirical perception experiments showed that only the multimodal model could account for the observed behaviour of human listeners. Another finding was that the data's fit was best when the model's prior probabilities were set to a low value (p.17f). They assume that this effect might be related to the high specificity of phonetic adaptation (according to specific individual speakers or situations).

A significant body of work on computational models of first language acquisition based on the speech signal has been done in the ACORNS project[6]. Learning is based directly on a representation of the continuous speech signal. The main conceptual difference between the "whole-utterance" approaches developed in ACORNS and other investigations of speech segmentation (including the experiments described later in chapter 4) is that a segmentation is learned first. A method to learn unsegmented words directly from combinations of speaker independent spectral patterns is, for example, developed by van Segbroeck and Van hamme (2009). Their algorithm could also be used to learn phones instead of words in a different setting, they argue. The approach is semi-supervised as the word types and their respective number of occurrence for each utterance are input parameters for the simulations (p.1133). The speech representation used in these models is unsegmented and non-sequential. The ability of the model to recognise new instances is interpreted as evidence that segmentation is perhaps not needed.

2.3. Phonetic and phonological structure

In his introduction to a special issue of *Computational Linguistics* on computational phonology, Bird (1994, p.vi) lists four research themes in this discipline: "Formal

[6] http://www.acorns-project.org (accessed 2010)

reconstruction and language-theoretic results", "implementations", "automatic learning" and "interfacing to grammar and speech". These research themes lend themselves to computer simulation studies. Bird points out that the theoretic framework of *The Sound Pattern of English* by Chomsky and Halle is "directly implementable". Using computer simulations in phonology often relates to the central concepts of *generative phonology*. One fundamental idea in generative phonology is that all phonological description is essentially algorithmic in nature. Phonological processes are algorithmic, i.e. they are discrete, formal sets of rules which can be processed in a finite amount of time using a Turing machine or any equivalent mathematical or actual computing machine. This essentially algorithmic nature of phonological frameworks offers promising starting points for computer simulation experiments.

Various formal and mathematical frameworks have been applied to model phonological structures and simulate their emergence and change and other phonological processes. The emergence of phonological structure has been simulated, for example, using Game Theory; for example by Jäger (2008) or Zuidema and de Boer (2009). As an example with relevance to phonetics and phonology, Jäger (2008) presents a pilot study in which Game Theory is used to simulate language evolution. The simulation presented by Jäger (2008, p.415f) operates on a two dimensional $F_1 \times F_2$ vowel space with varying numbers of vowel categories. The signals are interpreted as "phonetic events" which are located in the vowel space. Signal transmission is not perfect, i.e. there is noise which distorts the original signals, potentially causing the receiver to perceive a different signal from the one the sender had originally intended to send. The evolving population in this model corresponds to the set of phonetic events. In each round of the simulation, a vowel category is picked randomly (the total number of categories remaining constant). A memory stores past production and perception events. The outcomes of the simulation with 3–9 vowel categories could be compared to existing vowel systems of natural languages, and thus, a potential framework for the simulation of language evolution is proposed.

2.3.1. Syllable Structure

Several simulation studies have been carried out in the framework of articulatory phonology (Browman and Goldstein, 1992). The central assumption in Articulatory Phonology is that the articulatory gesture is the basic phonological unit of speech production. Nam et al. (2009) describe a simulation within the framework of articulatory phonology to investigate general phenomena of syllable structure. In particular, they address the question of the apparent cross-language preference for CV syllables over VC syllables, i.e. they seek to determine the cause for the *markedness* of the VC-type syllable structures (Nam et al., 2009, p.299f). Their two-agent simulations involve a child agent and an adult agent, where the grammar is pre-defined for the adult agent and the

child agent "tunes" its (initially random) grammar according to interactions with the adult. The simulations show that CV structures are acquired faster than VC structures, irrespective of the target language of the adult agent. This agrees with findings in various natural languages (see Nam et al., 2009 for references). Their simulations allow two predictions which could not be tested due to the lack of available data. The first is that CV structures should appear earlier in a child's productions, even in languages, where such structures would be regarded as phonologically ill-formed. The second prediction is, that in children's early CCV productions the two consonants are often produced simultaneously. Such cases are predicted to be perceived by adults and, more importantly, by adult transcribers as simple CV syllables (Nam et al., 2009, p.320).

Recently, Tilsen (2011) reported on an investigation of the effect of metrical regularity on speech production. A production study is performed in which participants produced a series of non-word utterances with either regular strong-weak syllable patterns "sww.sww.sww.sww" or irregular patterns "sww.wsw.wsw.sww" (where "s" represents a strong and "w" a weak syllable). The results indicate that metrical similarity of the elements in an utterance influences word durations and production errors (p.203f). To interpret this result, Tilsen (2011, p.206) implemented a dynamical model based on the framework of articulatory phonology. A simulation was carried out which showed higher activation for regular metrical patterns over irregular patterns. Tilsen (2011, p.211) notes: "If planning system activation is considered to correspond to a strength of activation in working memory, and if selection/execution occurs more quickly and accurately when word-stress systems are more strongly activated, then the selection of elements in the regular sequence is predicted to occur more quickly than in the irregular sequence." The results of the production study and the simulation are interpreted as arguments against serial production models which cannot account for the observed interference between the sequentially produced utterance items.

2.4. Speech perception and production

This section gives a short overview of computational studies on (models of) speech perception and production. There is a large body of research on this topic which stretches over various disciplines beyond phonetics and phonology. Especially speech perception has been studied extensively in the fields of psycholinguistics and computational psychology (cf. Massaro, 1996). The identification of reliable cues for phonetic segments and categories or their acoustic correlates in a speech signal is a research topic with a long history. Results from studies on ASR and speech perception have mutually informed the two corresponding fields of research. The potential of using computers in experimental phonetics and phonology has been recognised early. Cooper (1961) reviews the characteristics of available speech synthesizers and discusses their potential use for different research topics in phonetics. He points out that speech synthesizers

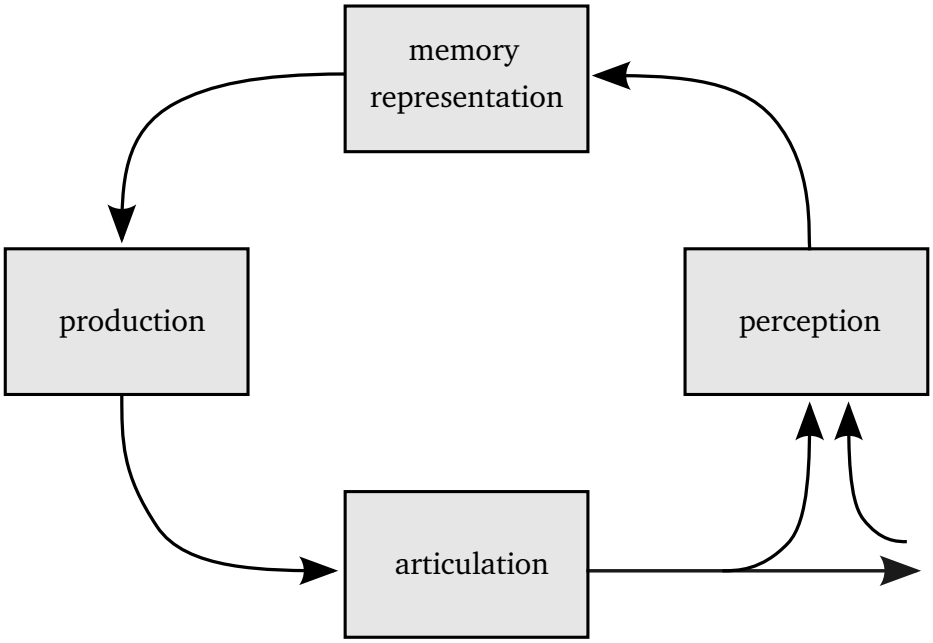


Figure 2.1.: Illustration of the perception–production loop

could be employed to identify the "significant carriers of information" in a speech signal "by *manipulating* the spectrum and *hearing* the result" (p.4). Damper (1998, p.263) points out that simulation experiments can be used, for example, to identify the cues in a speech signal which are important to categorisation but which "may not be obvious". However, almost half a century after Cooper's (1961) presentation, Johnson (2008, p.421) writes that speech is "an inherently messy and ambiguous physical (acoustic or gestural) signal." He argues that speech is perceived in a context-sensitive manner based on equivalences on multiple levels. Thus, a radical solution to the non-invariance problem might be that there is no absolute invariance at the acoustic or phonetic level:

> "Invariance exists in a relational sense only, to be tested 'ceteris paribus', that is in the same context. Absolute invariance is a property of the perceptual-cognitive process induced by linguistic competence rather than a property of the physical form."
> (Fant, 2004, p.B-35)

Similarly, Hawkins (2003, p.386) emphasizes that phonetic categories should not be regarded as absolute but instead as relational like phonological categories. She outlines a model of speech perception which incorporates fine phonetic detail of speech items. Phonetic categories, she argues, are context-sensitive. Examples of simulation studies on speech perception include work by Johnson (1997a) or Kello and Plaut (2004).

It has been observed in empirical studies that speech production is influenced by the prosodic and phonetic details of previously perceived speech. Lewandowski (2012,

p.166), for example, reports on convergence effects in dialogues between native and non-native speakers. She found that in such situations both non-native and native speakers converge on the phonetic level towards their interlocutor. Such a close relation between speech production and perception is the basis for theoretical models like the motor theory of speech perception (e.g. Liberman and Mattingly, 1985) or articulatory phonology (e.g. Browman and Goldstein, 1992). Consequently, speech perception and production are often modelled as closely linked phenomena, being parts of a *perception–production loop* (e.g. Wedel, 2004a, 2007). Figure 2.1 shows an illustration of the perception–production feedback loop on a very simplified level. The rectangles symbolise different processing modules. Incoming edges represent input to a module and outgoing edges represent a module's output. The details vary in the models proposed by different authors. The most important aspect of the graph is its topological structure consisting of a closed cycle. The number of the individual modules, their functions (and labels) and their interconnections are specific to the particular models, as can be seen in the three models discussed in sections 2.4.2, 2.4.3 and 2.4.4. The distinction between one module for production and one for articulation in figure 2.1 illustrates the often made distinction between a linguistic composition or the planning of an utterance and its realisation on the articulatory level. Note that only one part of the loop can be observed directly: the result of the articulation module — i.e. the acoustic speech signal and (to some extent) the articulatory movements that generate it. In terms of figure 2.1, this is only the lower right part of the graph (this interface is visualized by the outgoing and the incoming edges connecting to the outside-world). The other parts cannot be observed directly by objective measurements. Thus, computer simulation experiments provide an ideal tool in this field of research.

As it is impossible to give a complete overview, this section focuses on two specific topics. First, section 2.4.1 presents some simulation experiments investigating the *perceptual magnet effect*. Then, three examples of computational models implementing the *perception–production loop* are presented in 2.4.2, 2.4.3 and 2.4.4.

Articulation is discussed in section 2.6 of this chapter. The topic of speech segmentation, which is closely related to speech perception, is addressed in detail in chapter 4 (cf. especially section 4.1 on page 84, which gives an overview of related work in that domain). A further production model, the CSM, is discussed in detail in chapter 6.

2.4.1. The perceptual magnet effect

The term *perceptual magnet effect* refers to an observation that phonetic perception is altered by a listener's language exposure. Perceived distances of sounds appear to be decreased in the surrounding of phonetic prototypes (cf. Kuhl and Iverson, 1995, 122f). Prototypes are ideal abstract mental representations of sounds which are judged by listeners to be the best instances of a specific phonetic category. It has been observed

that they function as non-linear attractors, or perceptual magnets in phonetic space such that the perceived similarity between sounds is increased if they are closer to a prototype.

The observation of the perceptual magnet effect has triggered a large body of research including several computational modelling and simulation studies. Lacerda (1995) is a notable early example, who proposed an exemplar-theoretic model to account for the perceptual magnet effect (cf. section 6.1). An account of the perceptual magnet effect based on artificial neural networks (ANNs) is proposed by Guenther and Gjaja (1996) (cf. section 2.4.2 below, for more details on the model). They claim that the effect is a consequence of the formation of non-uniform neural maps in the auditory system based on language experience (p.1111f). As in Lacerda's (1995) model, there is no need for explicitly defined prototypes. Additionally, the proposed model does not require labelled exemplars. The main aspect of the model is a self-organised neural map which is trained in an unsupervised fashion through exposure to input data. The authors point out:

> "An explanation that does not require linguistic category knowledge probably better captures the phenomenon when one considers that the effect is already present by six months of age, likely before infants have developed awareness of phonemes as linguistic units [...]"
> (Guenther and Gjaja, 1996, p.1112)

A series of computer simulation experiments was carried out to compare the performance of the implemented model with the results of empirical studies on the perceptual magnet effect. A first experiment investigates the discrimination of American English phonemic categories /r/ and /l/. A second simulation investigates representations of vowel distributions and a third simulation investigates in more detail the warping of the perceptual space near a category centre (p.1115ff). The perceptual magnet effect is explained as a consequence of asymmetric activation patterns in the neural map which in turn reflect the non-uniform distributions of the input (training) stimuli.

Feldman and Griffiths (2007) and Feldman et al. (2009a) present a Bayesian model[7] to simulate the *perceptual magnet effect*. Feldman et al. (2009a, p.757) propose a model which is based on the idea that speech perception can be described as "a kind of optimal statistical inference". They test a two-category implementation of their model on Kuhl and Iverson's (1995) empirical data with thirteen equally spaced stimuli in a psychoacoustic $F_1 \times F_2$ space between /i/ and /e/. Feldman et al. (2009a, p.760) report results for a simulation carried out on a one-dimensional representation of the stimuli and point out that "the simulation is a concrete demonstration that the model can reproduce empirical data on the perceptual magnet effect quantitatively as well as qualitatively using a reasonable set of parameters, supporting the viability of this rational account." The simulations allowed Feldman et al. (2009a) to investigate the influence of different model parameters, like the prior probabilities of categories, the variability of categories

[7] See section 3.2.

or the amount of present noise (p.766). Examining the influence of such parameters empirically would be extremely difficult in some cases. Prior probabilities or variances of phonetic categories, for example, depend on the listeners' mental representations of their language(s). This cannot be controlled by the experimenter in perception tests. However, using computer simulation experiments allows for arbitrary parameter combinations. Later, Shi et al. (2010, p.449) extend that model by incorporating principles from Exemplar Theory.

2.4.2. DIVA

The model proposed by Guenther and Gjaja (1996) belongs to the larger framework of the DIVA model. The name DIVA is short for "Directions (in an orosensory space) Into Velocities of Articulators" (Guenther, 1994b, p.44). It is a neural network model of speech acquisition and production based on a perception–production loop. It has been applied in a number of computer simulation studies on various aspects of speech production and perception.

Speech acquisition is addressed by Guenther (1994b). The model learns input–output mappings during a babbling phase. A central part of the model is the *speech sound map* which encodes different speech sounds. The speech sound map mediates between auditory and motor speech representations and plays therefore a "pivotal role in imitation learning in the DIVA model" (Guenther and Vladusich, 2012, p.412). The term "speech sound" is used to refer to phonemes, syllables or words (cf. Guenther, 2006, p.351).

Acquisition is simulated by random generations of articulatory movements and learning the corresponding mappings from phonetics to sensory feedback to articulations (Guenther, 1994b, p.45f). Speech recognition based on the acoustic speech signal was not implemented in this earlier version of the DIVA model.

Speech perception and sound categorisation is addressed by Guenther et al. (2004). They present two simulation experiments based on which they conclude that learning of phonetic categories "leads to a decrease in auditory cortical activity while processing prototypical members of the category" (p.16). This decrease can be interpreted either as a decrease in the number of cells involved in the processing of prototypical speech items, or as a decrease in processing time necessary for the processing of such prototypes. Guenther et al. (2004) argue that it is more important to detect between-category differences than within-category differences. Accordingly, the learning process implemented in their simulation experiments results in higher sensitivity to stimuli close to category boundaries (confirming the results of the similar study reported earlier by Guenther and Gjaja, 1996).

Auditory expectations of production targets are produced for the speaker's own voice based on stored exemplars from other speakers. In addition, somatosensory expectations are produced for the current speech sound.

> “These expectations are based on prior successful attempts to produce the sound, though we envision the possibility that some aspects of the somatosensory targets might be learned by infants when they view a speaker (e.g., by storing the movement of the lips for a bilabial).”
> (Guenther et al., 2006, p. 8)

Guenther (2006) reports on an investigation of speech production and acquisition within the DIVA framework. One problem with DIVA is its modelling of acquisition: The first stage is babbling, i.e. learning to produce speech sounds; the second stage is training on sample speech sounds, which corresponds to an infant's exposure to her native language. The question arises whether it should not be rather the case that perception precedes production? This is at least the argument put forth by Duran et al. (2010b) (see chapter 4).

Guenther and Vladusich (2012) present a further extension of DIVA as a speech production model. The speech sound map neurons are compared to mirror neurons found in the F5 region of the premotor cortex of monkeys. In imitation learning, *error maps* are used to detect discrepancies between “the intended and the actual sensory states” (p.409f). This means, that in this stage, the infant can map from a speech sound produced by another person (usually an adult), to a speech sound that forms the intended target for imitation of the perceived adult speech sound. One of the key properties which the speech sound map neurons are assumed to share with mirror neurons are that they “spike” during active production and passive perception of certain motor actions. This is also an assumption in the CSM, where the stored context-sequence is used in both production and perception. Incoming auditory feedback is compared against a target from the speech sound map. If the incoming signal is not within the “target region”, error neurons will be activated. I.e. the target regions inhibit excitation of the error neurons. Only if incoming signals are outside of the target region, they will activate error neurons which initiate corrective motor commands. Once the model has learned correct feedforward commands, there will be no activation of the error map and no feedback will be needed in order to produce a speech sound correctly. After the model has learned to produce speech sounds relying on feedforward controls only, the error map can still be activated, if, for example, an external perturbation occurs. “Due to neural transmission delays and the delay between muscle activation and the resulting movement, these corrective commands will be delayed by approximately 75–100 ms relative to the onset of an unexpected perturbation” (p.414).

In summary, DIVA is a neural network model of speech production and perception. It implements a perception–production loop. Each of its components is explicitly associated with a specific anatomical location of the human brain (Guenther and Vladusich, 2012, p.412). It has been continuously refined and expanded since its introduction, and it has been applied to investigate various research questions.

2.4.3. Elija

Howard and Huckvale (2005) present an imitation-based model for human speech acquisition which was later named *Elija* (Howard and Messum, 2011). The model acquires speech sounds by simulating a *babbling* phase during which articulatory gestures are performed randomly with the results being stored in memory. This method allows the system to learn an inverse mapping from acoustics to articulatory gestures. The simulations use an articulatory speech synthesizer based on Maeda's (1990) work. The success of the training phase was evaluated by re-synthesising input speech and listening to the results and visual inspections of the corresponding spectrograms.

The principle of imitation is fundamental to later work on the model by Howard and Messum (2011). They argue that "learning to pronounce can be seen as the child's solution of the 'correspondence problem' between speech sounds he hears and speech sounds he makes" (p.86)[8]. The most significant aspect of this implementation of Elija, distinguishing it from other similar models, is that it interacts with a caregiver. The model learns from the caregiver's reformulations of its output (p.88). Howard and Messum (2011, p.94) stress that Elija does not imitate the caregiver. Therefore, they describe the approach as "non-imitative". The motor patterns created by Elija can be compared to the gestural scores assumed in models of articulatory phonology.

2.4.4. ACT

Kröger et al. (2007) present a neural model of sensori-motor speech development. A central part of the neuro-computational model is a 3D articulatory-acoustic model of the vocal tract developed by Birkholz (2005). The model learns neural representations for somatosensory–to–motor relations, quasi-stationary auditory–to–motor relations, dynamic auditory–to–motor relations, and for lexical–to–sensory and lexical–to–motor relations. There are four corresponding learning stages in the model: A silent articulation stage, followed by quasi-stationary vocalic articulation and a stage of articulatory proto-gestures. Finally, a linguistic stage of learning where simple single words. Each learning phase builds on the results of the previous one.

A further elaboration of the model for speech production and perception is presented by Kröger et al. (2009). The production part starts from a phonemic representation of a speech item, which is assumed to be generated by higher-level linguistic modules that are not described by the model (p.794). The term "speech item" is used to refer to speech sounds, syllables, words or entire utterances. The motor plan used in the model is, however, syllable-based. The assembling of a motor plan for infrequent syllables from sub-syllabic units is currently not implemented in the model[9]. The lower level primary

[8] The correspondence problem is also a central topic of the simulations presented by Duran et al. (2010b).
[9] This might be achieved by incorporating an approach similar to the *Multi-level model* developed by Walsh et al. (2010).

motor map comprises 10 articulatory parameters, each encoded by 2 neurons with complementary activation. The higher level motor plan map for V, CV, and VC-items consists of 11 neurons and depends on the length of the currently produced utterance. An *auditory-to-motor pathway* (dorsal stream) is included in the model, which "is capable of modeling categorical perception of speech sounds and is capable of modeling differences in categorical perception of vowels and consonants" (p.796). Training consists of two basic stages: *babbling* and *imitation* (p.798). During babbling, the phonemic map is not active. The knowledge gained from the babbling state is "language independent general phonetic knowledge". This stage is further divided into a stage for learning proto-vocalic states and a stage for learning proto-syllabic CV and VC-states. In vowel imitation training, a set of 500 training vowels is generated (representing realisations by different speakers) and mapped into the model's babbling vowel space (p.801). This corresponds to an implicit speaker normalisation (cf. Johnson, 1997a). For the sake of simplicity, imitation training is performed on isolated vowels and not on connected speech. The simulation also implements two sources of coarticulation effects: Speech gestures are not encoded in all details on the motor plan level and the exact coordination of articulators is controlled by the motor execution module. This results in context-dependent execution. Additionally, gesture specifications can vary even on the level of the motor plan (e.g. in non-relevant phonemic features). Temporal aspects are not included in the model, since it models only single syllable productions (or single sounds). In larger utterances, processing delays must be considered, e.g. as in the DIVA model (p.806).

The major differences between DIVA and ACT, according to Kröger et al. (2009, p.797), are: (1) DIVA does not separate motor planning and motor execution, (2) DIVA does not introduce a *phonetic map* as it is assumed in ACT, i.e. a self-organising map of speech items between sensory, motor and phonemic representation. Moreover, DIVA does not claim bidirectional mappings between phonemic, sensory, and motor representations.

According to Kröger et al. (2010, p.24), ACT emphasises the following aspects: First, articulatory gestures are considered the basic units of sensorymotor control of speech articulation. Second, the phonemic-to-motor state map and the phonemic-to-sensory state map are implemented as self-organising maps (SOMs). Finally, a *motor planning module* is introduced as an alternative neural pathway for infrequent speech items. Two stages of motor planning and execution are distinguished which lead to two different representation of articulatory information: A motor plan representation defines motor plans by specifying gestural scores, and a primary motor representation directly controls the execution of articulation by defining activations of the respective functional groups of muscles.

The architectures of ACT, as well as that of DIVA and Elija, correspond basically to the scheme outlined in figure 2.1 on page 52. The number of modules, their functions and mutual connections, however, differ in the three models as discussed above.

2.5. Language change and evolution

> “It is languages which adapt to a generic brain, and not the contrary.”
> (Oudeyer, 2006, p.62)

In an early article on the use of computers in linguistics, Lamb (1961, p.411) speculates about potential applications of computer simulations to study language change by applying “certain proposed general principles of linguistic change or certain kinds of external influences” to a specific language model and analysing its development over time. I refer to the development of an existing linguistic system over time as “language change”. The term “language evolution” is used here for computational studies on the origins of language and the human speech faculty. Although the term “evolution informally refers to any change through time”, as Baker (2008, Footnote 1) points out, models of language change build on an existing linguistic system (grammar), while models of language evolution do not presuppose an existing grammar. Within the framework of language change research, phonetics and phonology are subsumed under “lexical change” (e.g. Baker, 2008, p.290) or “sound change”.

The origin of the human language faculty and language is definitely inaccessible to experimental phonetics and laboratory phonology. Probably the only possibility to address this issue and test different theories by means of experiments are computer simulations. Boë et al. (2007), for example, compare the vocal tracts of humans with those of Neanderthals. They use vocal tract models to investigate whether the Neanderthals were physiologically capable of producing vowels like humans. The Neanderthal’s vocal tract was modelled. Based on this model, a speech synthesizer was used to simulate the potential Neanderthal vowel space. The results indicate that the Neanderthal’s vocal tract would have supported the production of human-like vowels.

The simulations by Boersma and Hamann (2008, p.250f), cited above, successfully replicated the diachronic development of the Polish three sibilant system from medieval /ʃ sʲ s/ to present-day /ʂ ɕ s̪/. Wedel (2009) discusses the hypothesis that a similarity-biased speech communication error can cause phonological change and entrenchment, and that this hypothesis can be analogously applied to morphophonological and morphological phenomena.

Emergent phonology in the development of sound systems is investigated by de Boer (1998) by means of simulation experiments. He emphasises that models which explicitly depend on the optimisation of some auditory or articulatory parameters are not suitable, since “humans do not do any optimisation when learning a sound system of a language. They just imitate their parents (and their peers). In fact, they imitate their parents much better than would be required for successful communication” (p.2). He presents a multi-agent simulation and refers to the interaction between the individual agents as “imitation games”.

Oudeyer (2006, p.61) reports on a computer simulation based on an imitation game similar to de Boer’s simulation which, however, uses syllables and not static vowels.

The syllables are based on "a shared repertoire of phonemes (imagined as the result of an earlier game [...])." This side note is very important as it reveals a fundamental assumption which is made in most models of speech perception–production and acquisition: the hierarchically structured language system which is at the lowest level based on combinatorial patterns of atomic units like phonemes. Oudeyer (2006, p.60f) argues that other simulations model language change given an existing linguistic system and not "the evolution of language in general" where a complex linguistic system emerges without a prior existing one.

Nettle (1999) uses computer simulations based on Social Impact Theory to investigate language change under varying conditions of social structure and acquisition biases. The constant source of newly introduced variation in his simulations is *imperfect learning*, although he acknowledges that this is a simplification and that there are other factors in real language change processes (p.97). The simulations reported by Nettle (1999, p.103) presuppose discrete differences between alternative variants and do not consider gradual variation. A set of individuals (agents) is arranged in a closed social-network (i.e. there are no boundaries). Every agent goes through 5 "life stages". It learns a variety p or q according to their corresponding impact functions (Nettle, 1999, p.102):

$$\hat{\imath}_p = b_p N_p^a [\Sigma(s_i/d_i^2)/N_p] \tag{2.2}$$

$$\hat{\imath}_q = b_q N_q^a [\Sigma(s_i/d_i^2)/N_q] \tag{2.3}$$

where b_p is an acquisition bias towards variety p, s_i is the social status of agent i with variant p, d_i is the social distance to the learner, N_p is the total number of individuals with variety p, and a is a constant parameter which controls the "persuasiveness" of the majority variety (or: "the importance learners give to conformity"; p.115). Under the assumption of imperfect learning, the general expectation would be that the majority variant is always learned and any new variants are simply outnumbered and quickly dropped. Without differentiation in functional bias or social status and a value of $0.5 < a \leq 1$, the majority variety will indeed remain the dominant variety within the population. With a value of $0 < a \leq 0.5$, however, the outcome after a number of iterations is always a split population, with one group having variety p and the other variety q. Both cases do not reflect the situation of real language change, where a new variety q emerges and finally replaces an old variety p completely. This, however, can be achieved by manipulating the parameters for social and functional selection.

The simulations by Nettle also illustrate that, while methods from other disciplines can be used to investigate phonetic phenomena, the underlying assumptions might require modifications. Nettle discusses such necessary modifications. The first one is a model parameter which was introduced in order to model heterogeneous distributions within one community (e.g. differing opinions). In order to model language change, however, Nettle (1999, p.102) argues that the parameter has to be changed since the assumption

is not that diverging opinions are the usual case but that in the long run a language community always reaches almost total agreement (note, that this is by definition the case within a speech / language community). A second possible modification to the sociologically motivated model parameters is a quadratic distance function. This is supposed to provide a realistic model of "the effect of geographical distance on the diffusion of cultural traits" (p.110).

2.6. Articulation, coarticulation and speech synthesis

The prototypical simulation in all speech sciences is the artificial production of speech: *speech synthesis*. As noted earlier, I focus on computer simulations of human speech production, perception, its internal representation and historical development, both for the individual speaker and for multi-generation speech communities.

In 1961, Cooper noted that "speech synthesizers are not new" (p.3). In his review of speech synthesizers, presented at the fourth International Congress of Phonetic Sciences (ICPhS), he points out two uses for speech synthesizers in phonetic research: (1) using speech synthesizers as "informants" to study the relationship between acoustic features and perceptual events, and (2) using them as models of the vocal tract to study the relationship between articulatory parameters and the corresponding acoustic results (p.11). Since this thesis is concerned with computer simulation experiments, the first mentioned use case of a speech synthesizer as a tool in computer aided perception experiments is, therefore, not of primary interest here. The second case, however, is a good example of computer simulations in phonetic research. Articulatory speech synthesizers, in particular, have been used for investigations of human speech production. The synthesizers reviewed by Cooper (1961) are, however, all special hardware devices. Only as a note on "newer developments" are speech synthesis systems mentioned which are implemented as computer programs. In the same year, Kelly and Gerstman (1961) presented one of the first computer based speech synthesizers.

Some of the earliest examples of computer simulations of human speech production are cases of articulatory speech synthesis. Although I will not consider speech synthesis in general in this thesis, I do include work on articulatory speech synthesis in this overview. If we define simulations as imitation of the behaviour of a complex system, and if the system we are interested in is the human speaker then today's industrial state-of-the art speech synthesizers do not qualify as simulations. If anything, they simulate the output of human speech production, but are not overly concerned about the actual processes involved in it. However, articulatory speech synthesis aims at producing speech sounds by imitating the dynamic changes inside the vocal tract and reproducing the acoustic consequences accordingly. An often referenced work is the articulatory

computer simulation implemented by Henke (1966)[10]. Ladefoged (1964) discussed (analogue) speech synthesis systems and their potential applications in the phonetic study of language. The "computer simulations" Ladefoged has in mind are of the type that specifies the cross-section area at a specific number of segments (cf. Maeda, 1990). He does not seem to have anticipated the type of 3-dimensional vocal tract models available today for articulatory speech synthesis (cf. Birkholz, 2005). Lisker et al. (1962, p.82f) discuss the necessity to test the formal descriptions in phonology and propose doing so by using speech synthesizers.

Articulatory speech synthesis can be useful in investigating the speech *perception–production loop*, and in particular the *correspondence problem* in language acquisition or articulatory inversion. Using a simulation with an articulatory speech synthesizer allows for the investigation of how the correspondence between articulation and the acoustic consequences can be learned beginning from a babbling phase (as discussed in section 2.4).

Toda and Maeda (2006, p.2) use speech synthesis to investigate "quantal aspects of non-anterior fricative sibilants". They use an acoustic three-tube model which simulates a constriction in the vocal tract, the size and location of which is a parameter of the model. The lengths of the model's cavities are systematically varied. Based on the simulation results, they conclude that there are at least four quantal regions which may shift according to the speaker's anatomy. They assume that this is one factor responsible for an individual's articulatory strategy (p.8).

2.7. Summary

This chapter provided a detailed, yet incomplete overview of a wide range of topics in the study of spoken language. From abstract phonological systems and their development over multiple-generations of speakers to an individual's articulatory habits — computer simulation experiments have been successfully applied to investigate the applicability of existing models and to develop new models. As mentioned in the introduction, the work presented in this chapter covers a great variety of research disciplines. The historical precursors to modern speech synthesis systems were tightly associated with musical instruments. This provided not only the necessary tools but also the theoretical framework to investigate the production of speech sounds. As this chapter shows, today's investigations of human speech need to incorporate theory and methodology from neuroscience, computational psychology, speech signal processing or artificial intelligence and machine learning.

This present overview shows how very different approaches can be applied in the study of the same subject. The following chapter provides another overview of related

[10]Authors citing the dissertation by Henke (1966) include for example: Mermelstein (1969); Guenther (1994a); Kröger (1998); de Boer and Fitch (2010).

work with a focus on methodological issues and frameworks used in computer simulation experiments. It will show how very different subjects can be studied with the same methodological approaches.

3. Methods

This chapter gives an overview of the most frequently used methods and formal frameworks in computer simulation experiments in the fields of phonetics and phonology. These are mainly methods of computational modelling and machine learning. Most of the referenced work discussed here is also discussed in the previous chapter. However, this chapter does not focus on the 'what' and 'why' aspects of computational simulations but on the question 'how?'.

I cannot go into the details of machine learning theory in this thesis. This would be far beyond its scope. This chapter is not meant to give a fully detailed overview of each particular aspect of the discussed methods. The full theoretical and practical aspects of the presented methods are beyond the scope of this thesis. I will restrict the level of detail to cover only the relevant aspects necessary to put this thesis and the discussed related work into context.

The first sections focus on methods of machine learning. These are particularly relevant to multi-agent simulations, were the individual agents in a simulation build up their own internal knowledge representations (e.g. vowel systems). This chapter is structured as follows: Section 3.1 gives a very brief overview of basic aspects of artificial intelligence and machine learning which are particularly relevant for the issues discussed in this thesis. Bayesian learning is discussed in section 3.2. Artificial neural networks and connectionist models are discussed in section 3.3. The principle of self-organisation and the self-organising map (SOM) are discussed in section 3.4. The approach of using artificial data is briefly discussed in section 3.5. Finally, this chapter is concluded by a section on evaluation in 3.6.

Data clustering and its evaluation is discussed in detail in section 5.2 of chapter 5.

3.1. Artificial intelligence and machine learning

Two basic types of learning are distinguished in the NLP and speech processing literature, and in machine learning in general: *supervised learning* and *unsupervised learning* (cf. Bishop, 2006, p.3). For the purpose of the work discussed here, *supervised learning* can be defined informally as a process of constructing an approximation of a relation which is implicitly given by a set of input–output pairs and the generalisation of this relation to the entire input domain. In supervised training, thus, a learning method is presented with input data as well as the corresponding expected output (e. g. a recorded

speech signal along with its time-aligned phonetic transcription). A learning method is called *unsupervised*, on the other hand, if it receives only input data without explicit specification of the corresponding output which it should produce (cf. Bishop, 2006, p.3; Kiang, 2002, p.308). In the scope of speech segmentation, unsupervised methods are also referred to as "text independent" (e. g. Aversano et al., 2001) or "blind" (e. g. Sharma and Mammone, 1996b) segmentation.

Apart from *supervised* and *unsupervised* learning, there are also other types of learning. One such method, which is neither strictly supervised nor unsupervised is *reinforcement learning* where the optimal output is not given explicitly but has to be found by the learning method based on an interactive process (cf. Bishop, 2006, p.3). Daland and Pierrehumbert (2011, p.131) point out that the distinction between supervised and unsupervised learning is not always clear for "'bootstrapping' models, in which some preexisting knowledge is leveraged to solve a different problem." We (i.e. Duran et al., 2010b, p.137) have previously used the term *semi-supervised* to refer to approaches which rely in part on available a priori knowledge — but not about the solution to the actual task.

Unsupervised methods are, in my view, of primary interest for computer simulation experiments in phonetics and phonology which include some learning component. Language acquisition cannot be plausibly modelled with a supervised approach. Infants do not receive the types of information that can be found in speech and text corpora: there is no explicit information about the segment boundaries, no explicit information about the metrical structure of the language, about the phonemes, or the lexicon. It might be argued that formal language *learning*, in contrast to language *acquisition*, is in some respect a supervised learning scenario. In general, supervised learning methods are cognitively implausible. Cairns et al. (1997, p.144) argue that supervised training "does not correspond to human development" since "explicit training is not normally a part of human development". Murre and Goebel (1996, p.75) speak of "real-time learning" and point out that this is the most frequently used mode of learning — i.e. the absence of a teacher who gives explicit instructions about "what to learn and when". Therefore, the focus in this present thesis is on non-supervised learning methods and their applications in various computer simulation experiments.

3.2. Bayesian learning

This section gives a short overview of a specific form of statistical learning based on *Bayesian statistics* — which I will refer to as *Bayesian learning* — and their applications in the investigation and computational simulation of human speech production and perception. From a cognitive perspective there are some cases in which humans seem to exploit statistical learning, for example in language acquisition. Some frameworks in cognitive science or psycholinguistics seek to model this on the basis of Bayesian

learning. An overview of Bayesian learning in cognitive sciences is given, for example, by Griffiths et al. (2008) or Perfors et al. (2011). Bayesian learning can, in general, be used to model problem solving mechanisms which involve inductive inference, but also to model processes of making "inherently underconstrained inferences from impoverished data in an uncertain world" (Griffiths et al., 2008, p.61).

Bayesian learning is often used to learn models iteratively from sequentially ordered data. It is often applied in the domain of speech technology. Pernkopf et al. (2009), for example, use Bayesian networks for the automatic classification of speech sounds for automatic speech recognition, or Goubanova (2003) uses Bayesian modelling of vowel duration for text-to-speech synthesis, to name just two examples out of a large body of research. However, there are only few Bayesian learning frameworks in the fields of phonetics and phonology and simulations of human speech perception and production.

One essential aspect of Bayesian statistics is the assumption of *prior beliefs* (Manning and Schütze, 1999, p.54f). These are presupposed prior probabilities of the set of events under consideration. The beliefs are then updated iteratively according to observed events using *Bayes' theorem*, the basic formulation of which is given in equation 3.1 (adopted from Manning and Schütze, 1999, p.43):

$$P(B|A) = \frac{P(B \cap A)}{P(A)} = \frac{P(A|B)P(B)}{P(A)} \tag{3.1}$$

In order to determine the probability of B (e.g. a model or a specific hypothesis) given A (e.g. an observed event), $P(A)$ is considered the *prior probability* of A. Initially $P(A)$ must be set to some a priori probability distribution—the *prior probability*. Once a new event is observed, the a posteriori probability distribution $P(B|A)$ is calculated. This *posterior probability* then constitutes the new prior for the next iteration. This method of *Bayesian updating* is used to learn models iteratively from a sequence of events. The term $P(A|B)$ is called *likelihood*.

An advantage of Bayesian models is, as Griffiths et al. (2008, p.62) point out, that they can be used to investigate issues of prior knowledge without the need to resort to *innateness*:

> "Crucially, these models do not require that prior knowledge be innate. Bayesian inference in hierarchical probabilistic models can explain how abstract prior knowledge may itself be learned from data and then put to use to guide learning in subsequent tasks and new environments."
> (Griffiths et al., 2008, p.62)

Examples of Bayesian learning in speech science include, for example: work by Goldwater et al. (2009), who use Dirichlet processes to model unsupervised speech segmentation. Kleinschmidt and Jaeger (2011b,a) use Bayesian belief updating to simulate phonetic category adaptation. Ellis (2008) uses Bayesian learning to model the phonological process of generating surface forms from underlying representations. Morley

(2008) uses Bayesian learning to investigate the general question of how grammar arises by modelling phonological systems. Feldman and Griffiths (2007) and Feldman et al. (2009a) present a Bayesian model to simulate the *perceptual magnet effect*. Later, Shi et al. (2010) present an exemplar-theoretic implementation of that model (cf. section 2.4.1). Feldman et al. (2009b) also use a Bayesian framework to simulate the emergence of phonetic categories based on segmented speech. Norris and McQueen (2008) present SHORTLIST B, an updated version of an earlier model for the simulation of continuous speech recognition which was based on principles of Bayesian learning.

3.3. Artificial Neural Networks and Connectionist Models

Artificial neural networks (ANN), also simply called *neural networks* or *connectionist models* (cf. Murre and Goebel, 1996, p.49; Manning and Schütze, 1999, p.603; Kohonen, 2001, p.73f; Kiang, 2002, p.303; Bishop, 2006, p.226), are mathematical models of networks of connected simple units. The units are commonly referred to as *neurons* according to biological neural networks in the brain. Specific weights are associated with each connection in the network. These connections are established along with the states of the individual neurons in a learning or training process based on some input data. Each neuron receives input from an external source and/or from a number of other neurons and produces an output signal which is fed to other neurons (and probably back to the producing neuron itself, depending on the particular topology of the network). The structure of an ANN can be represented by a highly connected weighted directed graph. The neurons correspond to the nodes of directed graphs and the connection strengths between neurons correspond to the weights associated with the directed edges connecting the nodes. There are, in general, *excitatory* and *inhibitory* connections between neurons, corresponding to positive and negative edge weights, respectively (Kiang, 2002, p.305). Different network topologies and learning algorithms, i.e. rules for systematic modifications of the connection weights (Kiang, 2002, p.320), lead to different specific types of artificial neural networks. One such specific type of ANN — the SOM — is discussed in more detail in section 3.4.1 below. However, discussing the full spectrum of ANN architectures in any detail is beyond the scope of this thesis. Neural networks are referred to as *connectionist models* primarily in the fields of psycholinguistics, computational psychology and other brain sciences (cf. Murre and Goebel, 1996, p.49f).

There are two possible interpretations of ANNs: one is that of an actual model of (parts of) the brain and the other is that of sophisticated "practical inventions for new components and technologies" — Kohonen (2001, p.71) points out that, in contrast to contemporary ANNs, the early models of the 1940s and 1950s were primarily interpreted

in the first way, but that both "motives [...] may have been around all the time". Perfors et al. (2011, p.314) point out that the existence of both excitatory and inhibitory connections, for example, are biologically implausible. Thinking of biological neural activities as digital 0/1 impulses is mentioned by Kohonen (2001, p.73) as "one of the oldest misinterpretations of the physiological observations". He continues and emphasizes that there are many different types of neurons, many different chemical transmitters, that information processing in the brain is not discrete but continuous and that it takes place at different time scales. In essence, the nervous system cannot be compared to a programmable digital computer — "the brain circuits are not automata" (p.79).

Keeping these facts in mind, ANNs are still useful tools in computer simulation experiments. For example, ANNs are in particular "[...] capable of modeling extremely complex functions, especially for problems that are nonlinear" (Kiang, 2002, p.314). Mitra et al. (2012, p.2275) demonstrate how ANNs can be used to learn non-linear mappings between multi-dimensional input and output spaces in the articulatory–acoustic domain. Kohonen (2001, p.81) lists some cases when ANNs should be considered. Among these are: noisy and ill-defined data, modelling of collective, nonlinear properties of the input data and the emergence of intelligent information processing functions — pointing out that ANNs can create abstractions in a completely unsupervised fashion. These cases apply to natural speech.

Examples of studies using ANN-based methodologies include work by Guenther (1994b), Guenther and Gjaja (1996), Kello and Plaut (2004), Kröger et al. (2009) or Mitra et al. (2012), to name just some of them[1]. An ANN is used by Kello and Plaut (2004, p.2354f) in a simulation experiment on phonological development. The model parameters are learned from a speech corpus which contains acoustic speech signals and corresponding articulographic measurements. Without incorporation of a priori knowledge about the segmental and phonological structure, the model learns mappings from articulatory to acoustic representations. A vector representation was used for both acoustic and articulatory data. The input vectors contained articulatory information of three consecutive time windows and the corresponding output vectors of the model represent acoustic power spectra (p.2358). Guenther and Gjaja (1996, p.1111ff) investigate the *perceptual magnet effect* with a neuro-computational model (DIVA) simulating the self-organised formation of neural maps. One simulation replicated findings from earlier empirical studies on the discrimination of English /r/ and/l/ categories by adult Japanese listeners (p.1115). Another simulation addressed differences in vowel perception by Swedish, English and Japanese infants (p.1116). A third simulation addressed perceptual discrimination of synthetic vowel stimuli by adult listeners (p.1117). In

[1] See section 2.4.2 for more details on the DIVA framework and the Guenther (1994b) study; section 2.4.4 for more details on the Kröger et al. (2009) study within the ACT framework. See section 5.1 for more details on the studies by Kello and Plaut (2004) and Mitra et al. (2012).

another series of simulation studies on neurocomputational models of speech perception and production (within the "ACT" framework), Kröger et al. (2009) employed ANNs to simulate the acquisition of linguistic knowledge. An example of a simulation experiment with symbolic input data is the study by Cairns et al. (1997, p.132f) on speech segmentation in which a neural network is trained on segmental data derived from a speech corpus (see section 4.1.1).

3.4. Self-organisation

Strictly speaking, *self-organisation* is not a simulation method. It is a form of behaviour of systems. The general idea is that if a system is let to develop on its own, it will become more ordered over time. Kauffman (1993, p.xiii) refers to self-organisation as "the spontaneous emergence of order". Oudeyer (2006, p.32f) characterises self-organisation as a "property" of complex systems in which ordered structures emerge on a macroscopic scale which cannot be readily explained by the properties of the system's constituting microscopic entities. The phenomenon of self-organisation in complex systems has recently received increasing attention in a variety of scientific disciplines. In linguistics, and particularly in phonetics and phonology, self-organisation receives increasing attention as it seems to provide a powerful means of describing various processes. de Boer (1998, p.2) writes: "Emergent global behaviour from local interaction is called *self-organisation* [...]". The emergence and change of speech sound systems, for example, is described by de Boer and some other authors as a process of self-organisation. Wedel (2011) provides an overview of computational simulations based on principles of self-organisation in the field of phonology. With respect to the apparent ubiquity of self-organisation in natural and social systems (p.142), he remarks:

> "...if we found that self-organizational mechanisms played no role in the emergence of any observed language patterns, our burden would be to explain why not."
> (Wedel, 2011, p.135)

3.4.1. Self-Organising Maps

Self-organisation, as a general principle, is most often implemented in computer simulation experiments via *self-organising maps* (SOM), sometimes also called *Kohonen maps*[2]). Self-organising maps are a kind of artificial neural network (ANN). They are discussed separately in this section as they are particularly relevant for computer simulations in phonetics and phonology. Kohonen defines his self-organising maps as follows:

[2] For example by Manning and Schütze (1999, p.527)

> “The SOM may be described formally as a nonlinear, ordered, smooth mapping of high-dimensional input data manifolds onto the elements of a regular, low-dimensional array.”
> (Kohonen, 2001, p.106)

Kohonen (2001, p.XI) quotes Cottrell et al. for pointing out that the SOM is “surprisingly resistant to a complete mathematical study”. Therefore, I will take this as a warning and not attempt at doing so in this very short discussion of the SOM — in addition to the fact that a detailed and formal discussion of the SOM is beyond the scope of this thesis. After a general introduction of the basic structure and its properties, the SOM will be discussed as a tool for computer simulation experiments in phonetics and phonology.

Self-organising maps provide a framework and general purpose software tool for the implementation of self-organised, i.e. unsupervised learning in computer simulations and various applications of machine learning. In particular, SOMs provide a sophisticated method for clustering and visualisation (Kohonen, 2001, p.XI). A useful property of the SOM for that purpose is that it preserves the topology and hierarchical structure of high-dimensional input data (Murre and Goebel, 1996, p.74; Kiang, 2002, p.312). They can be used as a tool for the abstract (i.e. simplified) visualisation of high-dimensional data (Kohonen, 2001, p.106). They are applied in a broad variety of disciplines[3]. Kiang (2002, p.312) identifies the primary applications of the SOM as: “clustering, pattern recognition, and various optimization problems”.

The outline of the basic algorithm for updating the SOM according to a list of input samples can be stated as shown in pseudo-code in algorithm 3.1 (following Kohonen, 2001, p.105ff). The following describes the basic idea of self-organising maps, their data structures and the updating or learning procedure. A self-organising map, defined over an input data space X, comprises the following elements:

- a finite, ordered set of “models” (usually corresponding to an array structure with a dimensionality which is lower than that of the input space X): $M = \{m_i \in X\}$
- a distance measure d on points in the input space: $d : X \times X \rightarrow \mathbb{R}$
- a mapping from the input space to the array of models: $f : X \rightarrow M$

The set M of models (or neurons) forms a lattice which can have a rectangular, hexagonal or irregular typology (Kohonen, 2001, p.110). Each node in M consists of a *model vector* $\vec{m}_i$ (also called *reference vector*), with an integer index $i \in \mathbb{N}$. A sublist of input samples which have been mapped to this node can be attached to each node.

[3] A bibliography of research articles, published in three parts, lists 7768 papers on self-organising maps or work from various research fields in which self-organising maps are applied. (cf. Kaski et al., 1998; Oja et al., 2002; Pöllä et al., 2009)

Algorithm 3.1 Basic SOM algorithm

```
Let X_i be a finite list of input samples x(t), with index t ∈ ℕ
for all x(t) in X_i do
    Map x(t) to the model m_c with the smallest distance from x(t)
    Copy x(t) into a sublist associated with m_c
end for
for all m_i ∈ M do
    Compute x̄_i for neighbourhood N_i associated with m_i
    Replace model m_i with x̄_i
end for
```

Two distance measures (with appropriate metrics) have to be defined for a SOM. The first one is the distance between points in the input space $d : X \times X \rightarrow \mathbb{R}$. The second required definition is a distance r within the grid of SOM nodes. This is required for updating the model vectors according to some input.

Let the *neighbourhood* $N_i \subset M$ be the set set of nodes within a certain radius in the grid around node i. Then, the *generalised set median* $\bar{x}_i$ of N_i is defined as:

$$\bar{x}_i = \arg\min_{x_i \in N_i} \sum_{x(t) \in N_i} d(x_i, x(t))$$

The learning process can be defined for example as: $m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)]$. The neighbourhood function $h_{ci}(t)$ is a smoothing kernel over the nodes of M. "For convergence, it is necessary that $h_{ci}(t) \rightarrow 0$ when $t \rightarrow \infty$" (Kohonen, 2001, p.111). Its purpose is to shrink the size of the neighbourhood with increasing learning time such that the learning process converges. Local "smoothing" due to co-activation of models in the neighbourhood of an activated model leads in the long run to global ordering (p.110).

The function $f : X \rightarrow M$ maps each input sample $\vec{x}$ to the model in M with the smallest distance from $\vec{x}$. It can be defined as:

$$f(x) = \arg\min_{\vec{m} \in M} \{d(\vec{x}, \vec{m})\} \quad .$$

In many applications, the Euclidean distance is used to define the best-matching node m_c as the one for which $||\vec{x} - \vec{m}_c|| = \min_i \{\vec{x} - \vec{m}_i\}$.

Examples

Kröger et al. (2009) use self-organising maps in a neurocomputational model of speech production and perception (see also section 2.4.4). Self-organising maps are implemented to represent various neural maps. For example, they interpret the central

"phonetic map" in their model as a neural map which corresponds to the mental syllabary (Kröger et al., 2009, p.795). They simulate human speech acquisition processes which acquire articulatory knowledge and establish correspondence links between production and perception (p.797f). More specifically, the model simulates *babbling* during which pre-linguistic speech knowledge is acquired. This knowledge is stored in self-organising maps which are interpreted as representing "general phonetic knowledge". In a second experiment, they simulate the production and perception of vowels and CV-syllables (p.802f). The model uses the knowledge acquired during the previous babbling training phase. This time, not only articulatory, somatosensory and phonetic information is processed. Phonemic information is added, represented by an additional SOM which was not used for babbling.

Miller et al. (2009, p.3f) do simulations on unsupervised speech segmentation[4] and use a SOM to "tokenize" a sequence of spectrograms. The SOM is trained on the speech data and the nodes of the trained SOM are used as labels in clustering the speech spectrograms. This effectively converts the quasi-continuous speech signal into a sequence of discrete symbols.

Miyazawa et al. (2011) present results of simulation experiments with self-organising maps on phoneme acquisition from recorded speech signals. They base their implementation on principles of human language acquisition and auditory processing (p.749f).

The representation of speech items with a SOM shares some similarities with exemplar theory and with prototype models (see section 6.1). All the stored models m_i of the SOM correspond to previously encountered input samples if the updating procedure takes one of the input samples, e.g. the median. However, unlike exemplar theory, not all input samples are explicitly stored in the SOM (beyond the immediately following update). In prototype models, on the other hand, the prototypes representing a specific category need not correspond to any actually encountered input item. The mapping is similar to prototype models if the updating procedure averages over the neighbourhood items and the resulting model does not correspond to an actual input sample.

3.5. Simulation experiments with synthetic data

Simulations of abstract systems (e.g. phonological studies on sound systems as those discussed in section 2.2) operate on data representations at a high level of abstraction. However, simulations more directly concerned with actual physical (or auditory) phenomena can also use relatively abstract data representations. In this case the data are often referred to as *synthetic*. The motifs for using synthetic data in simulation experiments are twofold: real data might be too complex/noisy for the purpose of the experiment or it might not be available at all. Mitra et al. (2012, p.2274), for example,

[4] See section 4.1.5 on page 97 for more details.

constructed a synthetic database with articulatory and acoustic information to train an ANN-based speech recognizer. Gestural information was constructed based on an X-ray microbeam speech database and a corresponding acoustic signal was synthesized. Mitra et al. (2012, p.2270) constructed a synthetic database due to the "lack of natural speech database containing gestural information".

Often, only annotations for phonetic segments are available in corpora with real articulographic information (e.g. in the MOCHA corpus, cf. section A.4). One notable exception is the Polish EMA corpus (see section A.1). For a study on gestural coordination in Polish obstruent–sonorant clusters reported by Bruni (2011, p.111f), a corpus was created with acoustic and articulatory measurements. In addition to word-level and phonetic segment-level annotations, gestural targets have also been annotated — although not the complete articulatory gestures.

3.5.1. Monte Carlo simulations

Monte Carlo simulations were originally developed with the aim of investigating neutron diffusion by Enrico Fermi and (independently) Stanisław Ulam to carry out statistical sampling automatically on a computer (Metropolis, 1987). Implemented as a programme on one of the first digital computers, these Monte Carlo calculations are the first computer simulation experiments. The basic idea of Monte Carlo simulations is statistical sampling by repeated calculations. Each individual calculation involves some probabilistic decisions. This procedure is repeated until a statistically valid result is achieved. This method is applied when a direct mathematical treatment of a problem is too difficult.

In linguistics, Monte Carlo simulations are used with generative models to approximate some unknown distribution by random sampling (cf. Goldwater et al., 2009, p.28f). Shi et al. (2010, p.443ff) use a Monte Carlo simulation to approximate Bayesian inference in a cognitive model of speech perception, and Lin (2005, p.136) uses a Monte Carlo simulation to approximate the Kullback-Leibler divergence (as a lexical distance measure) between sound distributions.

3.6. Evaluation measures

If computer simulations are understood as imitation of the behaviour of a system by its model (which is implemented as a computer programme given a parametrisation of the underlying model)[5], the question arises of how good this imitation is in comparison to the real system. Evaluations of computer simulation experiments are concerned with

[5] See terminology section 1.2 on page 37.

the *validity* or the *quality* of the simulation's observed behaviour or its produced output data.

This section provides general information on evaluation measures and methodologies. Evaluation of data clustering methods is discussed in section 5.2.4 on page 168. Evaluation of speech segmentation methods is discussed separately in detail in section 4.2 on page 99, where I also present some experiments investigating the properties of various evaluation measures for that domain (section 4.3).

The first distinction which can be made in the classification of evaluations is that between *subjective* and *objective* evaluations. Subjective evaluations of computer simulations in phonetics and phonology usually consist of perception tests. Some examples of such subjective measures where a system is evaluated by listening to its results or by manual annotations are cited in this thesis (e.g. Howard and Messum, 2011; or Miller et al., 2009). Objective measures compute a score based on the statistical properties of the output of a system and, optionally, an additional set of reference data. The term *evaluation measure* is used in this thesis exclusively for such objective evaluations. This is the focus of this section.

Two general types of objective evaluation methods can be distinguished: *external* and *internal evaluations*. Wedel (2004b, p.82) evaluates the results of his simulations by checking whether repeated runs confirm the previous outcomes — this can be considered an internal evaluation. An external evaluation compares a result with a given reference. For classification or segmentation tasks this reference is usually a manually created annotation of the investigated data. The notions of *error* and *accuracy* are important in this scenario. For the domain of data clustering, Dom (2002, p.138) notes on the usage of a human generated solution as a reference for external evaluations: "The idea is that the intended users of the algorithm would be quite happy if the algorithm had produced these classes as clusters."

Evaluation methods can also be distinguished as either *direct* or *indirect*. Indirect evaluations assess the performance of the overall system which uses an applied method as one of its data processing modules. For example, a speech segmentation method may be evaluated indirectly via the performance of an ASR system which builds on the segmenter's output.

It is common practice in NLP tasks to define a baseline condition against which the performance of a system is compared. This comparison against a baseline is important especially if the primary concern is improvement of the performance of a system (in terms of output accuracy, for example). A commonly used baseline in classification tasks is the assignment of the most frequent category label to each data item.

3.6.1. Basic terminology

The machine learning methods discussed in this thesis are usually applied to a data set such that they produce a number of decisions. Each of these individual decisions can be either correct or wrong (e.g. an assigned category label or the prediction of a segmentation boundary at a specific position). The following four quantities are important for the external evaluation of such methods in a variety of domains:

Notation	Meaning
TP	true positives
FP	false positives
FN	false negatives
TN	true negatives

The precise definition of these terms depends on the evaluated task. An example of a possible definition of these quantities is given in the next section in the context of a 2×2 contingency table which can be interpreted as a binary classification result. Section 4.2 discusses various possibilities of defining TP in the context of speech segmentation.

The reference data against which an external evaluation measure determines the quality or validity of a given result of a system is often referred to as *gold standard*, especially in NLP. The evaluated data, i.e. the output of a system, is referred to as *prediction* or *hypothesis*. The latter term will be used in the remainder of this section.

3.6.2. Confusion matrix and contingency table

The terms *contingency table* and *confusion matrix* are sometimes used interchangeably in the literature. For the purpose of this thesis, a contingency table is defined as a $n \times m$ matrix where the columns correspond to one random variable A and the rows to another random variable B, and each cell c_{ij}, with $0 < i \leq n$ and $0 < j \leq m$, contains the number of observations with value i of A and value j of B. A confusion matrix can be defined as a special case of a contingency table with $n = m$, where one random variable corresponds to the true categories of an underlying data set and the other variable corresponds to the predicted or hypothesized categories. Confusion matrices are often used to visualise and evaluate experimental results. In phonetics, for example, the confusion matrix is a standard tool to evaluate the performance of subjects in perception experiments — tabulating "stimulus" (variable A) versus "response" (variable B) frequencies. In this thesis, confusion matrices are arranged such that the columns correspond to the true categories (or classes) and the rows to the hypotheses generated by some simulation software (e.g. the results in section 5.3). Other computer simulation studies which use contingency tables to visualise and evaluate the results include Johnson (1997a, p.155),

Schweitzer (2010a, p.121f) or Feldman et al. (2009a, p.769). Contingency tables are useful evaluation tools for small numbers of columns and rows. Confusion matrices can become relatively complex for visual inspection with increasing dimensions.

For the case of binary classification, a contingency table can be used to define the quantities listed above in 3.6.1 (cf. Manning et al., 2009, p.155; Manning and Schütze, 1999, p.268):

	reference A	reference B
hypothesized A	TP	FP
hypothesized B	FN	TN

Thus, *true positives* are all items which are correctly hypothesized by some algorithm as being of class A (and analogously for the other quantities).

3.6.3. Precision, recall and F-measure

Precision and recall are two widely used evaluation measures in NLP, machine learning or information retrieval. Precision is the ratio of correct decisions out of all hypothesized decision, and recall is the ratio of made correct decisions out of all possible correct decision in the reference data. Based on the quantities defined in 3.6.1, they can be defined as follows:

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{3.2}$$

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{3.3}$$

As it is more convenient to quantify the quality of a generated solution by one number instead of two, the F-measure is often reported (van Rijsbergen, 1979, p.134). Based on precision and recall it is defined as:

$$F_\beta = \frac{(\beta^2 + 1) \cdot \text{precision} \cdot \text{recall}}{\beta^2\,\text{precision} + \text{recall}} \tag{3.4}$$

3.6.4. Entropy

Entropy is another often used evaluation measure. It is zero if there is no uncertainty about the outcome of an event. Formally, it is zero if the probabilities of all events but one are zero and this one having probability 1. Otherwise, entropy is always a positive

value (Shannon, 1948, p.394). It takes its maximum for the case that all probabilities are equal. This increases with the number of possible events. If the base 2 logarithm is used, the unit of entropy is the *bit*. The symbol H is usually used for entropy[6]. Note, that information-theoretic entropy is concerned with sequences of symbols. This might be a problem in certain scenarios where no discrete units can be properly defined.

One example of the use of entropy is the study by Tilsen (2011, p.187f)[7] who uses it as a means to quantify the metrical complexity of an utterance. He compares two regular metrical patterns (1) "sw.sw.sw.sw", and (2) "sww.sww.sww.sww" and two irregular patterns (3) "swww.sww.sw.sww", and (4) "sww.s.sw.swww.sw" (where "s" represents a strong/stressed syllable and "w" a weak/unstressed syllable). "A basic regularity metric that accords with our intuitive relative rankings of regularity" is obtained by combining 0th–2nd-order entropy rates for different regular and irregular metrical patterns. Tilsen (2011, p.188) defines these entropy rates as follows[8]:

$$H^0 = -\sum_i p(i) \log_2 p(i) \tag{3.5}$$

$$H^1 = -\sum_i p(i) \sum_j p(j|i) \log_2 p(j|i) \tag{3.6}$$

$$H^2 = -\sum_i p(i) \sum_j p(j|i) \sum_k p(k|ij) \log_2 p(k|ij) \quad , \tag{3.7}$$

where $p(i)$ is the probability of a syllable of type i, $p(j|i)$ is the conditional probability of a syllable of type j preceded by a syllable of type i and, accordingly, $p(k|ij)$ is the probability of a syllable of type k preceded by two syllables. These entropy rates are proposed as a quantitative measure of listener expectations and they are used to compare different metrical patterns.

3.7. Summary

This chapter complements chapter 2. While the previous chapter provides an overview of related work according to linguistic criteria, this chapter has a focus on frameworks employed in computer simulation studies and recurring methodological issues including evaluation methods and commonly used measures. A brief introduction into concepts of artificial intelligence and machine learning is followed by examples of simulation studies using a connectionist paradigm. The principle of self-organisation has been identified as a driving force underlying various natural and social systems. As formal mathematical

[6] The symbol H is interpreted either as the Latin capital h or the identically shaped Greek capital letter *eta*.
[7] Cf. page 51.
[8] My notation.

and analytical treatments of self-organising systems are particularly difficult, they lend themselves to computer simulation studies. Recently, ANNs enjoy revived popularity in the fields of speech processing and natural language processing (NLP). New methods developed there might inspire new lines of research and allow for new types of simulation experiments in phonetics and phonology.

Experiments

4. Boundariness and segmentation

> "Ja, selbst das ergebnislose Probieren vermag, wenn es richtig gedeutet wird, die wichtigsten Erkenntnisse zu liefern."
>
> *(Planck, 1931, p.26)*

This chapter presents a series of computer simulation experiments on unsupervised speech segmentation, a task infants are faced with in the earliest phases of language acquisition. Two lines of research are particularly relevant in this context: First, speech segmentation is an important research problem in automatic speech processing, for example concatenative speech synthesis or ASR. The segments in ASR are usually speech items at or below the phone/phoneme-level. In this domain they are typically considered to be known, and what is referred to as 'segmentation' is the task of time-alignment between a transcription and the speech signal (cf. Ljolje et al., 1997, p.308; Torre Toledano et al., 2003, p.617; Baghai-Ravary et al., 2009, p.2879). Alternatively, automatic segmentation performs such that the resulting segments do not correspond to linguistic units (for example, splitting the speech signal into equally sized chunks). The most probable label sequence is then determined in a second processing step by a trained classifier based on the sequence of these units. Approaches where neither the phonetic transcription nor the number of phonetic segments in an utterance are known, are called *blind segmentation*. An example of a segmentation method of this kind is provided by Sharma and Mammone (1996b,a).

Second, speech segmentation is investigated in psycholinguistics and cognitive science as a problem in the contexts of language acquisition and speech perception. The segments in cognitive models of speech segmentation are usually syllables or words (Massaro, 1996, p.87ff). The term 'speech segmentation' in this line of research refers to the task of finding word or syllable boundaries in the speech stream, while the underlying lexicon or phone inventory, i.e. the set of labels, might still be (partly) unknown (Brent, 1999a).

For this chapter, I assume a generalised view and regard speech segmentation as the task of partitioning a speech signal into non-overlapping smaller units. These units may refer to any linguistic level of abstraction, e.g. phones, syllables, words etc., or even to sub-phonemic units or entire sentences. The experiments presented here focus on *phonetic* segmentations, but no claim is made that the general observations cannot also be transferred to other levels of speech segmentation.

The structure of this chapter is as follows. In section 4.1 I give an overview of related work on human speech segmentation, focusing on computational models and simulations. Two main lines of research are addressed: work in the fields of psycholinguistics and cognitive science and work in the field of speech processing, especially ASR. In section 4.1.6, I discuss the concept of *boundariness* which is central to the experiments presented later in this chapter. Section 4.2 gives a theoretical overview of the evaluation problem and various evaluation measures. A traditional outline would place an evaluation section after the section describing the experimental methodology and setup. However, we (i.e. Duran et al.) observed that there are many inconsistencies and inaccuracies in the literature on speech segmentation simulations when it comes to evaluation. In addition, there is no agreement among researchers on which measure to use. This section, therefore, explicitly addresses the evaluation problem and in section 4.3 I present results of an experimental investigation comparing the properties of various evaluation measures. In section 4.4 I present segmentation experiments on continuous artificial data and on real speech which focus on boundariness and unsupervised learning of segmentation. Finally, section 4.5 summarises this chapter and puts the findings into the greater context of this thesis.

The basic outline and some preliminary results of the work discussed in this chapter have been presented by Duran et al. (2010a). An elaborate version with additional experiments has been published by Duran et al. (2010b) which lead to the formulation of the *correspondence-by-segmentation hypothesis* (see section 4.4). Section 4.2 is in part based on an unpublished paper by Duran et al. which originated from my experiments with various speech segmentation methods and their respective evaluations.

4.1. Related work on speech segmentation

This section gives an overview of computer simulation studies of human speech segmentation and related work.

Partitioning the continuous speech stream into a sequence of relevant discrete units is one of the fundamental problems for any kind of speech processing — be it in automatic speech processing tasks and applications or in models of (human) speech perception. Thus, segmenting the speech stream into linguistically relevant segments like syllables, words or phrases can be regarded as one of the fundamental human abilities. Without segmentation, utterances could not be analysed as sequences of individual speech items and would have to be stored as holistic units. More severely, abstraction of smaller speech items into types and re-combination of these types into new utterances would not be possible. Considering the fact that all linguistic theory and analysis of speech and language presupposes the existence of discrete units, recognisable as belonging to the same category despite discernible differences, speech segmentation can be seen as *the* fundamental property of language. It is the starting point of all phonetic and linguistic

analysis of speech.

The term "speech segmentation" refers to various related problems which differ in the size of what is actually considered a "segment" and in the type of input data which is considered as given. Most state-of-the-art approaches to the computational acquisition of linguistic knowledge in speech and natural language processing are statistical, supervised methods. These methods are trained on annotated data. Although these models can achieve a high accuracy (evaluated, again, against annotated data), they cannot serve as models for human speech and language acquisition. Humans do not receive explicit instructions in first language acquisition which are comparable to the detailed information that is extracted from manually annotated speech and text corpora. Infants are not explicitly told, for example, how many vowel phonemes there are in their respective language, or which patterns their formants have.

We seem to perceive speech as a sequence of discrete units which are all individually repeatable or replaceable, e.g. words or syllables. However, perceptual speech units must be distinguished from (lower level) auditory units and from (higher level) phonological units (cf. Massaro, 1996, p.87). There is only little work on the origin and the identity of the fundamental units of speech perception and production which does not presuppose linguistic knowledge. To this day, no completely convincing explanation exists for the question of, if and how fundamental units of speech perception/production emerge, and what such fundamental units might be. Computer simulations of human speech segmentation can help in gaining important insight into this complex and empirically hard to investigate matter. Related issues like the debate about innateness and the poverty of the stimulus argument[1] can be addressed, as demonstrated by Peukert (2008), by computationally investigating models of human speech segmentation and by

[1] The *poverty of the stimulus* argument is, in general terms, concerned with learnability of language. It rests on the claim that the information available in perceived speech (i.e. the experienced stimuli) is insufficient for learners to establish the complex linguistic knowledge underlying the observed surface forms. For a historical overview and a discussion of terminology see Thomas (2002). See also the more recent paper by Berwick et al. (2011, p.1209ff) who addressed the contribution of "innate, domain-specific factors" to the acquisition of linguistic knowledge. The authors conclude that despite many publications claiming the contrary, the argument still holds.

Usually, the poverty of the stimulus argument is limited to the investigation of syntax. However, Idsardi (2005, p.11) argues that "the paucity of extant phonological arguments for the poverty of the stimulus is a historical accident." He concludes: "Such arguments can be constructed [...] and are perhaps more telling in some ways than syntactic arguments, as there is no possibility of semantic boot-strapping for phonological rules." Of particular relevance within the scope of this present thesis is the work by Peukert (2008, e.g. p.6f; see also p. 90 of this thesis) who discusses the poverty of the stimulus argument and related work with respect to speech segmentation. He uses computer simulation experiments to investigate speech segmentation in first language acquisition. He states the primary goal of his thesis was a contribution to the discussion and refutation of the argument from the poverty of the stimulus: "Das primäre Ziel dieser Arbeit ist, zur Diskussion des Arguments der Reizarmut beizutragen und zu zeigen, dass die Wortsegmentierung im Erstspracherwerb zwar universalen, aber eher allgemein kognitiven Gesetzen gehorchen könnte".

investigating the information available in the speech signal.

There are still many open questions to fundamental problems in speech perception: Are there reliably identifiable and measurable cues to segment boundaries or boundariness within the acoustic signal (cf. section 2.4)? Do mental/phonetic/acoustic speech items correspond to the traditionally described phonetic or phonological speech segments like phones or phonemes? Or, are they perhaps shorter? Or longer? Are there different kinds of boundaries or different degrees of boundariness associated with different phone categories (e. g. stops, fricatives, diphthongs, glides, affricates etc.) and the transitions between them? Decades of acoustic speech analysis have worked on these problems, and no satisfactory solution is in near sight.

Port and Leary (2005, e.g. p.950) argue that language is essentially not symbolic and discrete but rather continuous and that the items in the lexicon do not consist of strings of symbols. Following this line of thought, the question has to be asked whether there are such definite boundaries in the speech stream like the ones conventionally presupposed by phonetics when segmenting a continuous speech stream into sequences of individual, discrete phones. Similarly, Hawkins (2003, p.386) argues that linguistic categories are not absolute but rather "dynamic, relational, i.e., context-sensitive, and plastic, i.e., labile." Her model of speech perception is based on exemplar-theoretic assumptions and treats representations of linguistic categories as implicit, private and "self-organizing and emergent" (p.388f). Segmentation of a speech signal is not a necessity of understanding speech but rather an emergent by-product of the listeners' individual internal representation and organisation of linguistic knowledge. To paraphrase Johnson (2008, p.419): "phonology is private."

Empirical observations show on the other hand that segmentation is "not merely an invention", as is pointed out by Johnson (1997b, p.104), for example. Kuhl (1987) gives an overview of several empirical studies on speech perception in early language acquisition. She discusses evidence that 6-month-old infants recognise different speech sound categories in different syllabic contexts spoken by different speakers. In one experiment, infants could recognise the similarity between syllables based on their initial consonant and distinguish the syllables /ma, mi, mu/ from /na, ni, nu/. Kuhl notes:

> "We know, then, that infants' representation of these syllables allows them to break the syllables down into some kind of 'parts' — ones that allow them to detect similarity at the beginnings of the syllables in spite of differences at the ends of syllables. At the very least, then, this ability must rely on a representation of units that allows *portions of syllables* to be isolated and compared across syllables."
> (Kuhl, 1987, p.352)

The remainder of this section first gives an overview of *symbolic-sequence approaches* from the fields of psycholinguistics and cognitive science. Then, *quasi-continuous approaches* are discussed which use speech processing methods for computational models of human speech segmentation.

4.1.1. Symbolic-sequence approaches

Most work on human speech segmentation in computational psychology, psycholinguistics or cognitive science takes on a symbolic sequence approach in which speech is represented as a string of discrete units. These units are, for example, phones, phonemes, syllables or words. In computational models, these units can be represented by different symbols or some kind of feature vectors. The problem of extracting these units from the continuous speech stream and recognising them as instances from a finite set of speech item classes is not addressed by these models.

Input for symbolic-sequence approaches

The input for these simulation and modelling studies is taken from text or speech corpora. Often, a phonemic transcription is obtained from the original corpus using a pronunciation dictionary. In some experiments this canonical pronunciation is then further processed in order to make "the result closer to oral French", or any other language (Xanthos, 2003, p.174; for similar approaches, see also Swingley, 2005, p.95f; Goldwater et al., 2009, p.29 or Daland and Pierrehumbert, 2011, p.133). Fleck (2008) evaluates her segmentation algorithm called *WordEnds* on input data taken from a number of corpora, and uses the term "dictionary transcriptions" to distinguish this kind of derived phonemic representation from real (narrow) phonetic transcriptions of actual utterances. Sometimes, orthographic input is used (e.g. Fleck, 2008). In such a case, the speech segmentation task can be compared to the word segmentation problem in NLP. However, word segmentation methods, e.g. for Chinese, most often apply a supervised learning approach, since this gives the best results. If the object of study is human speech segmentation, orthographic input can only be used with languages which have a (close to) phonemic orthography. Languages with a more etymological orthography cannot be plausibly investigated on orthographic input[2].

Standard practice in supervised NLP and machine learning tasks is to split the reference data into a training set and a test set. In psychologically plausible methods which do not learn in a supervised batch procedure but incrementally in an unsupervised fashion, like the one implemented by Xanthos (2003), such a split is not necessary.

Segmentation strategies

The proposed approaches to speech segmentation employ various types of basic strategies. Brent (1999a, p.296), who provides an overview of various computational speech

[2] Goldsmith and Xanthos (2009, p.9f), for example, simulate learning of phonological categories (see also section 2.2 on page 46 of this thesis). They use phonetically transcribed English and French input and, in addition, orthographic Finnish input, stressing that "written Finnish is notoriously close to a phonetic transcription". Similar claims could also be made about other languages with a (close-to) phonemic orthography, e.g. Croatian or Serbian.

segmentation models from this line of work, distinguishes between three basic classes of speech segmentation strategies: the *utterance-boundary strategy*, the *predictability strategy*, and the *word-recognition strategy*. A review article by Jusczyk (1999) discusses empirical evidence for a *boundary-cue strategy* in first language acquisition. Daland and Pierrehumbert (2011, p.122f) distinguish between *n-gram models* ("phonotactic" or "diphone and higher n-phone models"), *connectionist models* and *joint-optimisation models* which learn both segmentation and the lexicon.

Frank et al. (2007) use data obtained from an artificial language learning task with adults to evaluate a number of segmentation models implementing various segmentation strategies. The participants of the study had to listen for 15 minutes to synthesised samples from an artificial language. The words of that language were generated by concatenation of 18 syllables (p.282). They compare the performance of computational segmentation models which are based on the predictability strategy or the word-recognition strategy to the performance of the human subjects. The best fit to the empirical human data was found for a Bayesian model implementing a word-recognition strategy (cf. Goldwater et al., 2009).

Swingley (2005) follows Goodsitt et al. (1993) in distinguishing between *bracketing* and *clustering* strategies. Goodsitt et al. (1993, p.251) conclude that infants use combinations of various strategies to solve the segmentation problem. Thus, combinations of these basic strategies are used in some models and simulations. Examples of studies implementing one or more of these basic strategies are discussed in the remainder of this section. The following sections are not meant to provide definitive classifications of the cited models, however. Often, aspects of different segmentation strategies are employed in the same models. Especially the distinction between utterance-boundary and predictability strategies is rather subtle, and both might be described as some kinds of phonotactic strategies which are, in general, based on distributional patterns and frequencies of speech units.

Utterance-boundary strategy segmentation

Utterance-boundary strategy models of speech segmentation compare different levels of segmental representation. They are based on the assumptions that utterance boundaries are also word boundaries and that sequences immediately preceding utterance boundaries are in general similar to sequences immediately preceding word boundaries. Thus, boundaries on one given level of representation (e.g. the word-level) are hypothesised based on sequences of lower-level units (e.g. phonemes) which are characteristic of the endings (or beginnings) of the segments to be identified, and which are identified based on higher level segmentations (e.g. the utterance level). Figure 4.1 shows an illustration of the utterance boundary strategy and the involved levels of representation. An incremental, probabilistic implementation of the utterance boundary strategy is presented by Xanthos (2003). The input to the algorithm is a set of utterances which

are represented as sequences of phonemes (p.172). The simulation was performed and evaluated on transcribed child-directed French speech. The segmentation produced by the algorithm was evaluated against the original segmentation from the corpus computing *precision* and *recall* (cf. section 4.2 below). Xanthos (2003, p.178) points out that an advantage of the utterance-boundary strategy is that it relies on "perceptually salient features of the data rather than on the less obvious statistical properties used by predictability-based strategies". Another example is the *WordEnds* algorithm by Fleck (2008). It is an implementation of this type of segmentation strategy which also incorporates a morphological component to "fix" the segmentation produced by the actual segmenter[3]. She also reports results on seven different corpora with "dictionary transcriptions" as well as phonetic and orthographic representations of English, Arabic and Spanish speech (child directed speech as well as adult directed).

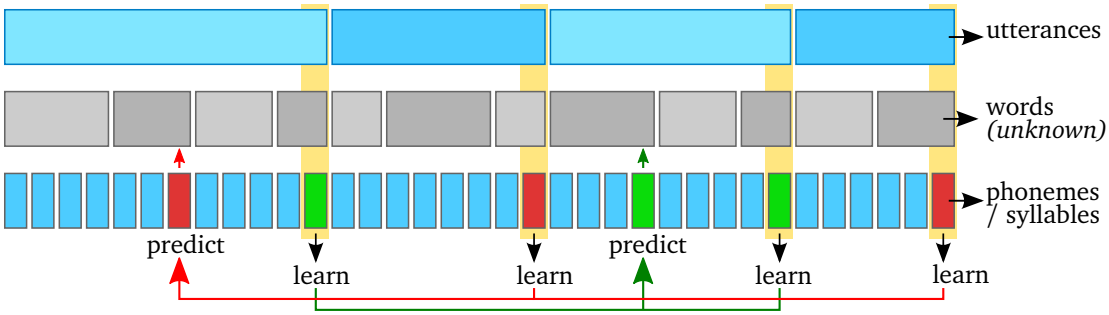


Figure 4.1.: Illustration of the principle of the utterance-boundary strategy of speech segmentation in symbolic sequence approaches. Blocks symbolise segments: utterances, words and phonemes/syllables. From patterns occurring at the end of utterances, a word boundary is predicted after the same patterns inside of an utterance. Red and green symbolise two different types of such observed patterns. As illustrated here, there might be word boundaries which cannot be detected with this strategy.

Predictability strategy segmentation

Predictability strategy models assume that guessing sequences of lower-level units (e.g. phonemes) is easier within segments than it is to predict sequences across segment boundaries. In a series of empirical experiments with seven- and eight-month-old infants, Goodsitt et al. (1993, p.229) investigated which type of strategy these infants apply in the segmentation of continuous speech. They consider two alternative speech segmentation strategies: "bracketing", which can be defined as an approach to segmentation based on the comparison of some properties of the transitions *between* items, and "clustering",

[3] Such multi-level or multi-module approaches often combine different segmentation strategies. Another example of this type is implemented, for example, by Peukert (2008).

which can be defined as an approach to segmentation based on the comparison of some properties *within* items[4]. They define "clustering" as aggregation of fundamental speech units with high transitional probabilities, placing segment boundaries at points of low transitional probabilities. Thus, clustering seeks to group similar items together, while bracketing seeks to separate dissimilar items. With respect to speech items, they consider syllables as fundamental units but point out that their argument also applies if the phone is taken as the fundamental unit (p.231).

A speech segmentation algorithm based on *mutual information* is presented by Swingley (2005, p.97). In contrast to much simulation work in this line of research which is based on phonemic/phonetic units, the fundamental unit of analysis in the work of Swingley (2005, p.93) is the syllable[5]. One of the experiments takes also resyllabification into account such that syllables might span word boundaries. The syllable boundaries are added by a probabilistic algorithm and the learning results are then evaluated over a 5-fold probabilistic processing of the base corpus data (Swingley, 2005, p.107f). Thus, while the input syllabification was in total ambiguous over the entire training data, the boundary decisions were categorical for each individual syllable instance. Ambiguous degrees of boundariness on a single instance are not considered by this approach (cf. section 4.1.6). The "clustering algorithm" described by Swingley (2005, p.97f) is not a method of cluster analysis as defined in section 5.2.4 on page 168. It is rather a two-way classifier that splits the syllable types into postulated words and non-words. Moreover, it merges some syllables into bi- and trisyllabic words.

Peukert (2008) developed a model of word segmentation in first language acquisition based on transitional probabilities (combined with an additional lexicon-based processing step). Provided that the input is a symbolic sequence, the author found that it does not matter whether the "unit of perception" consists of one or five phonemes. While only one third of the segmented words are correct, the most important result according to Peukert (2008, p.x) is that "the most frequent segmentations of the corpus happen to be lexical items". Peukert (2008, p.221) concludes from his simulation experiments that the argument of the poverty of the stimulus does not necessarily apply to speech segmentation, since all required information to solve the problem is available in the speech signal. The possible application of the symbolic segmentation algorithm to acoustic speech representations is also discussed by the author, but not implemented (Peukert, 2008, p.228f).

The phonotactic word segmentation model proposed by Daland and Pierrehumbert (2011, p.126) presupposes a two-stage account of speech processing in first language acquisition: a prelexical vs. a lexical stage. The authors emphasize the fact that, although infants do not possess "top-down" knowledge which facilitates word recognition, they

[4] In that sense, they use the term *clustering* differently from its meaning in the field of machine learning — see section 5.2 on page 159 for a discussion of data clustering methods.

[5] Massaro (1996, p.87ff) argues that the perceptual unit in speech perception is most likely the syllable.

exhibit robust word segmentation at the age of 6-12 months. At that age, infants supposedly have knowledge of only about 40-80 words.

Word-recognition strategy

Word-recognition strategy models attempt simultaneously to learn an inventory of speech segments (e.g. a lexicon of words) and to split a string of lower-level speech items based on recognition of the contents of this inventory. Based on the restriction that segments on one level of representation cannot overlap, segment boundaries are hypothesised where one known segment (i.e. a word) ends, and the next one begins. Usually, stretches of unrecognised speech between known segments are added as new words to the lexicon. The idea behind the application of this strategy is that a sufficient number of words are heard in isolation by the infant[6]. Daland and Pierrehumbert (2011, p.132), for example, speculate that a small lexicon based on words heard in isolation might be used in "semi-supervised learning" of speech segmentation.

A speech segmentation method implementing the word-recognition strategy is proposed by de Marcken (1996, p.108ff) within the broader context of unsupervised language acquisition. The algorithm learns a lexicon from unsegmented speech input and utilises the contents of the lexicon for segmentation. The learning procedure operates on a symbolic sequence of transcribed speech, which is automatically obtained from audio input by a separate system, external to the actual learning module. The work by de Marcken is one of the earliest to apply methods of ASR and machine learning to the problem of human language acquisition and speech segmentation.

A cognitively motivated segmentation method called *PARSER* is presented by Perruchet and Vinter (1998) and Perruchet and Peereman (2004). PARSER iteratively builds up a weighted mental lexicon (the "percept shaper", PS) through processes of reinforcement, interference and forgetting. At each time step an *initial parsing* is carried out by taking a randomly sized percept from the input stream. The weight of known units is increased when they are used to shape perception and new units are added to the weighted lexicon PS with an initial weight w'. Forgetting and interference processes repeatedly decrease the weights of the units in PS. Only units with an associated weight above a fixed threshold θ are considered for shaping perception.

Often cited word-recognition-strategy models of speech segmentation based on Bayesian learning (cf. section 3.2) have been developed by Goldwater et al. (2009). They compare the performance of their segmentation algorithms to other computational models on phonemic speech representations derived from the CHILDES corpus (p.29;

[6] Xanthos (2003, footnote 1) quotes a ratio of "about 9%" of isolated word utterances; Brent (1999a, p.300) estimates "the average frequency of isolated words in the speech of eight mothers to their infants (ages 9.5–12.5 months) to be about 7%, excluding interjections, onomatopoeia, social routines, and all words that did not also appear in multiword utterances (unpublished data)"; and Cairns et al. (1997, p.115) quote a ratio of 15% isolated words in child-directed utterances in the CHILDES corpus.

this is the same data as used by Brent, 1999b).

Transferring the word-recognition-strategy models to the lower level of phone or phonemic speech segmentation is not possible for several reasons. Learning word segmentation by lexicon building based on lower-level primitive units like syllables or phonemes is radically different from learning phone segmentation based on continuous speech data. The number of fundamental units in symbolic sequence approaches is finite (and usually very small in comparison to the target lexicon). On the acoustic level, on the other hand, variation is in any practical sense infinite, while the size of the target phoneme/phone inventory is small. Any two tokens are instances of the same word if they consist of the same sequence of phonemes while usually no two instances of one phoneme are identical on the acoustic level. The number of possible words is unrestricted in any natural language. Work in the Bayesian learning framework, for example, assumes an infinite vocabulary (Goldwater et al., 2009, p.44). The number of phonemes (or syllables), on the other hand, is relatively small. Although neither the total number of words nor the total number of phonemes should be restricted a priori in any plausible model of language acquisition, it is known that the number of words will continuously increase over time, while the number of phonemes will soon settle at a (more or less) stable, small value.

Boundary-cue strategy segmentation

Finally, *boundary-cue strategy* models attempt to identify reliable acoustic cues to speech segment boundaries. It has been hypothesised that infants have a disposition to search for and locate such cues in the continuous speech stream (Jusczyk, 1999, p.323ff).

Cairns et al. (1997, p.113) argue that a boundary-cue strategy is one possible way to avoid the chicken-and-egg problem of speech segmentation and recognition, where recognition of any stretch of speech as a meaningful unit depends on segmentation of the speech stream into units. In relying on cues to segment boundaries which are independent of the actual segments' identity, a bottom-up analysis of the speech input can be applied. They present a segmentation model which employs distributional information. The basis is a bottom-up ANN model using a predictability strategy on the sequence of phonetic segments. On top of that, pause and speaker-change information is added as an "unambiguous cue to a lexical boundary" (p.137). The simulation by Cairns et al. (1997, p.140) shows how sensitivity for metrical structure (i.e. the distinction between *strong* and *weak* syllables) can emerge from bottom-up learning of segmental context distributions. Importantly, the authors stress the fact that the model "has no need to rely on the *a priori* perceptual salience of strong syllables" to exhibit this property.

4.1.2. Quasi-continuous speech representations

The work discussed in the previous section on symbolic-sequence approaches is unsatisfactory from a purely phonetic point of view: the question is not addressed how phones are acquired from the continuous speech stream — a highly variable, often extremely noisy, continuous signal. It is generally accepted that learning how to segment the speech stream into a sequence of words is a fundamental task in language acquisition. Interestingly, the phoneme inventory is usually presupposed in symbolic sequence approaches and not further discussed.

It might be questioned whether the representation of speech as a sequence of phones or phonemes is appropriate at all — at least in the context of first language acquisition. The fundamental role of the phoneme has been questioned in studies on phonological awareness and illiteracy. Although infants show sensitivity to sub-syllabic patterns, the syllable is often assumed to be the fundamental unit in first language acquisition. It is sometimes argued that abstracting, for example, from prosodic information makes it possible to simulate the specific contribution of one specific source of information, e.g. phoneme transition probabilities in the case of Peukert's (2008, p.1f) simulations. Goldwater et al. (2009, p.29), on the other hand, point out that while symbolic representations of speech data lack many of the complexities of real speech, i.e. the acoustic signal, such representations do also lack many of potentially useful cues to speech segmentation like coarticulation patterns or prosody. Whether these differences balance out to make the task on both kinds of data comparably difficult remains an open question.

Presupposing a string of phone segments is more than just a convenience. It is a very strong assumption about the structure of the input to the learner and the available source of information. In addition, unsupervised methods that directly work on the speech signal to learn speech segmentation and to build an inventory of phone categories can be used to investigate whether the learned phones correspond to traditional phonetic segments.

4.1.3. Relation to automatic speech processing

There are two main types of speech segmentation which are discussed in the literature of automatic speech processing: (1) "audio diarization" and (2) "structural segmentation" (Ostendorf et al., 2008, p.61f).

Audio diarization can be described as non-linguistic or pre-linguistic segmentation. One typical example and subject to much research is automatic speaker diarization, where a speaker change is detected within a longer stream of speech. One particular problem in speech segmentation which is usually not of relevance in phonetics is *speech vs. non-speech* segmentation or separation. This kind of segmentation of an acoustic signal is just presupposed by any linguistic analysis which, by definition, discards any non-language sounds. However, with respect to language acquisition and the question

what knowledge the infant has to acquire about the world and what might be regarded as innate, this problem of segmenting auditory impressions into 'speech' and 'other' might be relevant to a computational model of speech perception or acquisition, just as it is relevant in speech technology. A speech technology perspective on the utterance boundary strategy, for example, reveals a very important assumption underlying such models: they rely on the ability to recognise utterance boundaries. At the very least, it is required that speech can be distinguished from non-speech in the acoustic input signal. Detecting utterance boundaries in symbolic sequence approaches is a trivial task since all symbols represent some atomic speech units (apart from some possible meta symbols, e.g. for 'silence' or 'utterance boundary'). Locating stretches of speech in an acoustic signal with all kinds of background noises, on the other hand, is not trivial.

Structural segmentation, then, could be described as the partitioning of a continuous speech stream into linguistically relevant units at any given level, e.g. phones, syllables, words or phrases. This is the type of speech segmentation phonetics is concerned with.

There is a long history of investigations of the problem of speech segmentation on a representation which is derived directly from the speech signal in the fields of ASR or concatenative speech synthesis. The methods developed there seem promising for phonetic research. Taking the speech signal as input instead of a symbolic sequence of a priori defined phonemic units seems more appropriate to the study of learning speech segmentation in first language acquisition. However, most of the approaches to segmentation in speech technology use methods of supervised learning to train a recogniser on a manual transcription aligned with an audio signal (cf. Morgan et al., 2004). Moreover, the utility of phonetic or phonemic transcriptions can be questioned in general in the domain of ASR. First, any manual transcription contains arbitrary subjective decisions influenced by the human labellers. The computational effort to derive an intermediate phonemic transcription from a speech signal is in most applications of ASR essentially unnecessary, since only the resulting sequence of words is of interest. Ostendorf (1999) questioned the suitability of the phoneme as a fundamental unit in ASR and proposed to move beyond the "beads-on-a-string" model of speech.

One important difference between various approaches to the segmentation problem from different engineering and research disciplines in relation to human speech perception is whether their respective focus is on the accuracy of the output or on the segmentation process itself. In speech technology applications, the primary focus is on the highest possible accuracy of a speech segmentation method (section 4.2 describes how the quality of a generated segmentation is measured). The (psycho-) linguistic or cognitive plausibility of the segmentation process itself is not relevant[7]. ASR, for example, can afford to discard purely linguistic criteria to segmentation if necessary and use whatever units work best with automatic classifiers. As soon as psycholinguis-

[7] The common practice to use hidden-Markov-models (HMMs) in ASR, for example, "has no relation to the properties of speech or to auditory processing", as Morgan et al. (2004, p.326) point out.

tic, phonetic or phonological plausibility is pursued, most of the current approaches to ASR cannot be used to simulate and investigate human speech perception without modifications.

In contrast to speech technology methods, the focus of simulations of human speech segmentation is not on the reproduction of a reference segmentation, but on the process itself of doing so. Of particular interest to the topic of computer simulation experiments in phonetic sciences are approaches which are "reaching over the gap" between speech technology and the study of human speech recognition (Scharenborg, 2007). Considering that the human brain is not a digital computer, ASR approaches to speech segmentation cannot, in principle, be readily used in models of human speech perception (cf. Kohonen, 1989, p.259; Kohonen, 2001, p.76). Particularly since ASR employs sophisticated techniques from machine learning, most often using supervised approaches. The acquisition of human speech segmentation abilities is not a supervised learning task in the sense of machine learning. The automatic segmenter is trained on segmented data. These reference segmentations are usually done manually. No such data is available in human first language acquisition (if it were, the task of automatic speech segmentation would trivially be solved by simply using those reliable cues from a speech signal).

4.1.4. Semi-supervised approaches

This section and the following give a brief overview of work on simulation experiments which investigate human speech segmentation based on a quasi-continuous representation derived directly from the digitised audio signal. The work on speech segmentation with quasi-continuous representations of the speech signal can be divided into two categories: semi-supervised approaches and unsupervised approaches. The term *semi-supervised* is used here to denote approaches which rely in part on human labels but not for the actual task to be solved. Methods which rely on human labels only for the initialisation of a model might also be classified as semi-supervised. An example of such a method has been developed by Ljolje et al. (1997). Another simulation study on speech segmentation based on the acoustic signal is presented by Johnson (1997b). One of the assumptions underlying his exemplar-theoretic model (cf. section 6.1 on page 217) of speech perception and segmentation is that the stored speech items, the "exemplars", are word-sized chunks which are created by primitive auditory scene analysis based on isolated word productions (p.104). A word recognition strategy is then assumed to be employed in speech perception and segmentation. One implementation of the model is trained to recognise eight words based on a representation composed of sequences of auditory spectra. New incoming items are compared against stored exemplars by comparing the sequential auditory representations one time-frame after the other. A segmentation of the exemplar under consideration seems to emerge implicitly from the activation patterns for the similarity computations of the various stored exemplars.

Depending on the current phonetic segment at a given point in time, the activation of exemplars with similar segments at that position increases in relation to the activation of exemplars with different segments. The activation for the input word ⟨cap⟩ is illustrated in figure 4.2. The three segments emerge from the rising activations of different words. First, the activation increases for exemplars which start with /k/, e.g. ⟨cap⟩, ⟨cat⟩ or ⟨keep⟩. Then, activation increases for exemplars with /æ/ in the second position, e.g. ⟨bat⟩, ⟨cap⟩ or ⟨cat⟩. Finally, near the end, activation increases for exemplars which end with /p/, e.g. ⟨bap⟩ or ⟨keep⟩[8]. Johnson (1997b, p.108) concludes that "[t]his analysis suggests that phonemes are defined in terms of subsets in the set of exemplars which have time-aligned similarities in their auditory/perceptual representations."

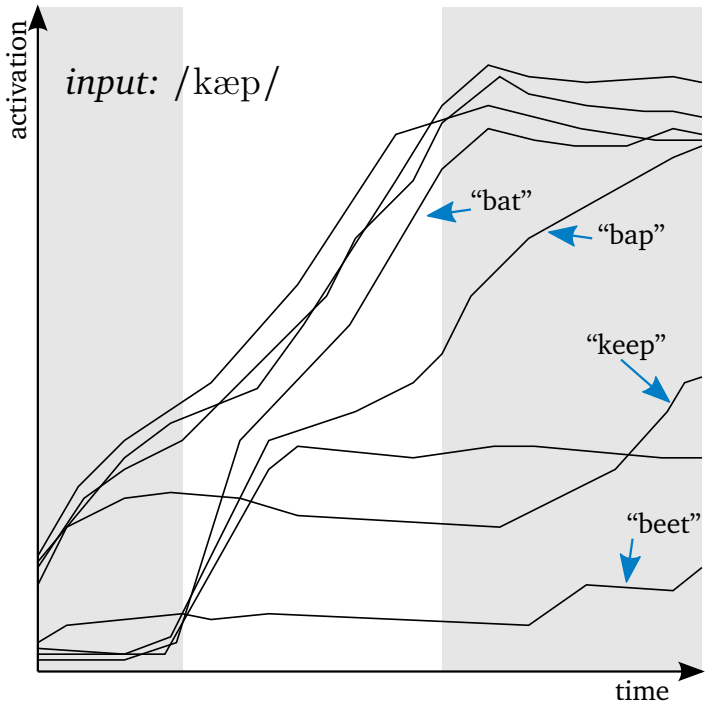


Figure 4.2.: Exemplar activation levels over time for the input word ⟨cap⟩ in Johnson's (1997b, p.107) model (figure adapted and simplified). The shaded areas depict (approximately) the three activation patterns according to the three phonetic segments of the input word.

Stouten et al. (2008) present a model that learns words from phone patterns based on the acoustic speech signal in an unsupervised way. The phone inventory and the acoustic-phonetic decoding are, however, defined a priori and not part of the learning task. Varadarajan et al. (2008) report on a method of unsupervised learning of phone-

[8] There are no units shown on the axes in figure 4.2 since they are not relevant for the purpose of illustrating the basic idea. Only the relative changes of the activation levels are important.

like units based on HMMs — a standard tool of modelling in speech technology, but cognitively not very plausible.

Improving ASR by incorporating knowledge about human speech perception is the motivation behind the simulation of van Segbroeck and Van hamme (2009, p.1125). They implemented an unsupervised language learning model which operates on acoustic patterns instead of a pre-defined set of phones or phonemes. The method is, however, not completely unsupervised since the word identities are known to the algorithm (p.1132f). Employing a kind of word-recognition strategy, the recognised words are used to partition a given utterance.

4.1.5. Unsupervised approaches

Computational models of speech segmentation with symbolic sequence approaches are common in cognitive sciences. The segments of interest in these models are words or proto-words. Although such approaches rely on an existing pre-segmentation of the speech stream with a pre-defined phone or syllable inventory, the origin of the fundamental inventory of primitive speech units and their identification in the continuous speech stream is usually not further investigated. Segmentation of speech is explained by distributional properties of lower-level units. This apparent chicken-and-egg problem is, however, sometimes acknowledged by the authors (cf. Cairns et al., 1997, p.112; or Swingley, 2005, p.87). As mentioned in the previous section, Johnson (1997b, p.104), for example, assumes a mechanism of "primitive auditory scene analysis" as the basis which provides a rough pre-segmentation of the speech stream for his speech perception and segmentation model.

Recently, a number of simulation studies have investigated phonetic speech segmentation based on quasi-continuous representations of the speech signal. The performance is often well below state-of-the-art methods from speech technology. Several methods have adapted algorithms which were originally designed for symbolic sequences. Gold and Scassellati (2006), for example, adapted the *minimum description length* algorithm, or Miller et al. (2009) adapted the *voting experts* algorithm. In this section, I briefly review some examples from this line of research.

Sharma and Mammone (1996b) present an unsupervised segmentation algorithm that partitions a speech signal without reference to any linguistic information about the utterance at hand (i.e. the total number or the sequence of phones). The method locates the segment boundaries in the speech signal according to the estimated "optimal number of sub-word segments". They illustrate their method on the test word "Manish" for which the algorithm computes an optimal number of six segments (p.3f). Using a database of thirty speakers saying "Alan Capone", they found that a speaker verification system based on the "blind" segmentation algorithm outperforms a system which relies on phonetic transcriptions (Sharma and Mammone, 1996a, p.96). The aim of their work

is not the investigation of phonetic speech segmentation. By not using a pre-defined phonetic inventory, the system is supposed to be able to handle an unrestricted, language independent vocabulary (cf. Sharma and Mammone, 1996a, p.93f). However, this work is interesting for phonetics and phonology as an example of speech segmentation based on acoustic information without prior knowledge of the phone inventory.

Lin (2004, 2005) describes an approach for learning phones directly from the speech signal, in which phonetic features are automatically learned, but with mixed results; e.g., half of all [b]'s are labelled as nasal. Segmentation is not explicitly addressed. The problem of having to rely on a symbolic sequence representation of the speech stream is addressed by Gold and Scassellati (2006, p.1370f). They compute MFCC representations for a set of thirty utterances and apply a method based on minimum description length, previously used only in symbolic sequence approaches. They derive acoustic features from the speech signal, but do not detect and evaluate phone boundaries. Miller et al. (2009) carry out segmentation experiments based on automatically tokenized speech spectrogram data. Two experiments were performed. The first one on synthesised speech samples which replicate stimuli from an empirical study with 8-month-old infants (Saffran et al., 1996); and the second experiment was carried out on read speech from a recording of George Orwell's novel "1984". They do not automatically evaluate how well their method recognises boundaries between phones according to a given reference segmentation. Instead, the hypothesised boundaries are evaluated manually by checking whether they fall within a 13 ms tolerance around an "obvious break location" (which is informally defined as "the beginning and ending of words, as well as phoneme boundaries where the audio stream suddenly changes in intensity or frequency"; p.5).

Scharenborg et al. (2010) investigate unsupervised speech segmentation and present a method which uses information about acoustic change based on *maximum margin clustering*. While this approach is based on *cluster analysis*[9], what it implements is actually a *bracketing strategy* of speech segmentation, not a *clustering* strategy — using the terminology as defined in section 4.1.1. The maximum margin clustering will produce a boundary at any given location of the speech stream. A second processing step is therefore added to the algorithm to decide on the placement of hard segmentation boundaries based on a peak-picking algorithm of the gradually changing distances between the cluster centroids (p.1086f). This graded strength of evidence for segment boundaries, which is implicitly incorporated in many segmentation models, is addressed in the following section.

4.1.6. Boundariness

One important concept in this discussion of speech segmentation is the notion of *boundariness*. Wade et al. (2010, p.237) introduced the term boundariness to the field of

[9] See section 5.2 on page 159 for a further details on data clustering methods.

phonetics for the graded nature of segment boundaries across different levels of abstraction. This is a basic idea behind my experiments presented in this chapter.

Although the term boundariness is usually not used, the concept is an important aspect in most computational and theoretical approaches to speech segmentation. Most segmentation algorithms first define a continuous boundariness function over the speech sequence and then derive a binary distinction between boundary and non-boundary in a second processing step. The boundariness function corresponds to the amount of evidence for the presence of a segment boundary at a particular point in time. Scharenborg et al. (2010, p.1087), for example, use a pattern matching approach on the changing classifications of the maximum margin clustering pre-processor in combination with a peak-picking approach on the Euclidean distance between computed feature vectors; and the model proposed by Daland and Pierrehumbert (2011, p.135) bases its binary boundary decisions on the continuous-valued probability of a word boundary between any two phones.

Thus, speech segmentation algorithms can be thought of as compositions of two functions $b \circ \beta$, where b is a boundariness function which maps each point along the speech signal's time axis to a boundariness value and β is a function which maps the real boundariness values to a Boolean value indicating whether there is a segment boundary at that point or not.

4.2. Evaluation measures for speech segmentation

> "Phonetic transcriptions must always be viewed as inherently untrustworthy."
> (Port and Leary, 2005, p.938)

The above statement is intentionally provocative. Still, it is safe to state that evaluating a given speech segmentation is not a trivial task, be it automatically generated or manually annotated. The following properties are often stated as desired features of a measure for segmentation quality:

- it should be a metric,
- near misses should be penalised less than larger deviations from the reference,
- it should be independent of the size of the data set,
- it should be interpretable,
- it should take into account over-segmentation,
- it should take into account inherent fuzziness of the data — this is especially important in speech segmentation where segment boundaries cannot be located consistently and unambiguously even by trained experts.

Although these desired properties might appear to be well motivated, they are still, in fact, completely arbitrary and subjective, and depending on the specific task or application, the requirements on an evaluation measure vary.

The definition of "errors" is generally not independent of the actual task under consideration. The distinction between target segments and non-target segments might be relevant for some tasks. Adding some non-speech material to a speech segment in ASR might be less severe than placing the boundary too early and cutting some of the speech material off the segment. Sometimes, for example, hypothesised boundaries within stretches of silence are not counted as errors. Therefore, it could be argued that a boundary is not simply correct or wrong. Different types of errors are relevant for different applications.

4.2.1. Terminology and basic quantities

The following symbols and notations are used in this section to discuss the various evaluation measures which have been proposed for the evaluation of speech segmentation:

Notation	*Meaning*
ref	the sequence of reference segment boundary indices
$hyp(b)$, or simply hyp	the sequence of hypothesised segment boundary indices (according to a boundariness function b)
$\|ref\|$	the total number of reference boundaries
$\|hyp\|$	the total number of hypothesised boundaries
$A_{a,b}$	the set of aligned segment boundaries from sequences a and b
TP	the number of true positives (or hits), i.e. the number of correctly hypothesised boundaries
$FP = \|hyp\| - TP$	the number of false positives (or insertions)
$FN = \|ref\| - TP$	the number of false negatives (or deletions)
$s(hyp(b), ref) \in \mathbb{R}$	an evaluation measure

4.2.2. Alignment

Many evaluation measures are based on the count of true positives *TP* and thus require a one-to-one mapping from (a subset of) *hyp* to (a subset of) *ref*. The usual approach is to align the hypothesised segmentation and the reference, and to select one-to-one links based on the distances between the hypothesised boundaries and the reference boundaries.

A hypothesised boundary is generally considered correct, and referred to as a "hit" or "detection", if its distance to a reference boundary is below a fixed threshold Δt, i.e. if it falls within the bounds of a defined tolerance window around the true boundary. Wesenick and Kipp (1996, p.132) report that evaluations of inter-annotator agreement in manual speech segmentation show 96% agreement for a tolerance window of $\Delta t < 20$ ms. A comparison of the agreement between two human annotators was also performed by Ljolje et al. (1997, p.309ff). For a tolerance of 20 ms, for example, the two human annotators agree on most boundary types, except for the boundaries between two vowels and boundaries between liquids followed by vowels, where they disagree in more than 26% of the cases (and, to a lesser degree, for boundaries between vowels and nasals and vowels followed by liquids). Based on such empirical observations on inter-annotator agreement, the window size is usually set to ±20 ms (e.g. Aversano et al., 2001; Dusan and Rabiner, 2006; Scharenborg et al., 2010), but results for a number of other window sizes have also been reported in the literature. For example, Ljolje et al. (1997, p.309) report percentages of boundary deviations of more than 10 ms, 20 ms, 30 ms, 40 ms and 50 ms; Torre Toledano et al. (2003, p.618f) report results for tolerances of 5, 10, 20 and 50 ms; and Qiao et al. (2008, p.3991f) evaluate their segmentation algorithm with tolerances of 20, 30 and 40 ms.

A potential alternative to the use of manually created gold-labels could be based on the approach presented by Baghai-Ravary et al. (2009). In order to evaluate phoneme boundaries in a speech corpus, they compared the results of several segmenters. The aim is to investigate which phoneme transitions are inherently ambiguous or "unpredictable" (in a sense, that any particular boundary location of this type is an arbitrary decision without an explicitly specifiable related objective physical cue in the signal). They point out that "phoneme boundaries are ill-defined objects and that each alignment system identifies them in its own characteristic way", and that such "[s]ystematic differences are not actually problematic or 'wrong'" (p.2882f). It is worth mentioning that even in this work, which claims to work "without reference to human labels", a manual phonemic transcription was used and the task of segmentation was that of aligning this sequence of phoneme symbols with the speech signal (as usual in ASR).

There are two types of possible errors, when a hypothesised segmentation is compared against a given reference. First, some reference boundaries might not have a matching hypothesised boundary, and, second, some hypothesised boundaries might not have a matching reference boundary. Reference boundaries that do not have a matching

hypothesised boundary are counted as false negatives and referred to as "deletions" or "misses". Hypothesised boundaries without a matching reference are counted as false positives and referred to as "insertions" or "false alarms". Aligning or matching an automatically obtained segmentation with a given reference is a non-trivial task, which often is not recognised as such, and underlying decisions are often not described or further discussed. However, implicit decisions can lead to different results. The problem of overlapping tolerance windows, for example, is not addressed by most authors as Räsänen et al. (2009) point out. The results can differ depending on how hypothesised boundaries within such overlapping regions are treated. Sometimes, hypothesised boundaries falling into a stretch of silence are counted as correct, irrespective of their actual location (e.g. Miller et al., 2009, p.5).

The resulting units of a segmentation method should be non-overlapping (Brent, 1999a; Scharenborg et al., 2010). If hypothesised and reference boundaries are matched sequentially one-by-one, boundaries should not be re-used in the evaluation if they are within the tolerance window of more than one reference boundary, or, if several hypothesised boundaries fall within the tolerance window of the same reference boundary. Scharenborg et al. (2010, p.1085) for example define the number of correctly hypothesised boundaries as: "the hypothesised boundaries that fell within the tolerance window distance from the ground truth boundaries", without mentioning the possible problem of overlapping windows or the multiple counting of hypothesised boundaries.

We (i.e. Duran et al.) propose the following criterion for declaring a pair of a hypothesised boundary h and a reference boundary r as a true positive:

Definition 4.1 *A hypothesised boundary $h \in hyp$ is assigned to a reference boundary $r \in ref$ and counted as a true positive iff:*

- *h is the closest boundary in hyp to r and*
- *r is the closest boundary in ref to h and*
- $|r - h| \leq \Delta t$

To break ties we adopt the convention that, if there are two equidistant boundaries, then only the earlier is eligible to be the closest.

This procedure is equivalent to the proposal of Räsänen et al. (2009) in its treatment of overlapping windows. In addition, it clearly defines how multiple hypothesised boundaries within the tolerance of the same reference boundary should be treated. The beginning and the end of the segmented sequence are trivially correct for all methods that place boundaries *into* a given input sequence. Thus, these two boundaries should be excluded from evaluation and don't need to be aligned. Nevertheless, they need to be considered in some cases: If a hypothesised boundary is actually closer to the beginning or to the end of the sequence than to its first or last (internal) reference boundary, it

should not be counted as a hit. In any other position, it would have to match the nearest boundary. Therefore, the sequence boundaries should be either aligned, or the alignment methods should ensure that a hypothesised boundary closer to the sequence boundary is not aligned with an internal reference boundary[10].

4.2.3. Over-segmentation rate

A very simple, often used global evaluation measure is the *over-segmentation rate* (OS). It is usually defined as:

$$\mathrm{OS} = \left(\frac{|hyp|}{|ref|} - 1 \right) \cdot 100 \tag{4.1}$$

If OS is negative, it is referred to as *under-segmentation* (e.g. by Scharenborg et al., 2010; Räsänen et al., 2009). It is a *global* measure as it does not take into account any local properties of the hypothesised segmentation but only the total number of segments.

4.2.4. Precision, recall and F-measure

Two of the most frequently used direct, objective segmentation quality measures are *precision* and *recall* (cf. Peukert, 2008, p.116). These measures are used for various tasks in computational linguistics and NLP (see also section 3.6.3 on page 77). The definition is based on the counts of generated decisions and a corresponding reference:

$$\text{precision} = \frac{TP}{TP + FP} = \frac{TP}{|hyp|} \tag{4.2}$$

$$\text{recall} = \frac{TP}{TP + FN} = \frac{TP}{|ref|} \tag{4.3}$$

These definitions based on true and false positives and negatives are general enough that they can be applied to many different problems — they are equivalent to the definitions 5.6 and 5.7 on page 172 for the data clustering domain. The crucial and, as discussed above, non-trivial part is the definition of the basic quantities *TP*, *FP* and *FN*.

Recall is often reported as a percentage, called *hit rate* (HR), *performance* (Aversano et al., 2001) or *correct detection rate* (Scharenborg et al., 2010). There are also other derived measures used, such as the *miss rate* (MR) or the *false alarm rate* (FA) with: $\mathrm{MR} = (1 - \text{recall}) \cdot 100$ and $\mathrm{FA} = 1 - \text{precision}$ (Pereiro Estevan et al., 2007). I do not further discuss any such derived measures. Their properties are fully determined by the properties of precision and recall.

[10] Peukert (2008, p.118f), for example, defines the total number of true boundaries as the total number of true segments (in his case word tokens) minus one.

Precision and recall measure complementary types of errors and usually both values are reported together. Moreover, having just one figure which considers all relevant error types is more desirable and it is also easier to compare. The most often used possibility to achieve this is the F-measure which combines precision and recall. Equation 5.9 on page 172 gives the definition of F for the domain of cluster analysis. For the evaluation of segmentation, the standard definition is used as follows:

$$F_\beta = \frac{(\beta^2 + 1) \cdot \text{precision} \cdot \text{recall}}{\beta^2 \,\text{precision} + \text{recall}} \tag{4.4}$$

where β is usually set to 1 (e.g. Räsänen et al., 2009; Kemp et al., 2000; Goldwater et al., 2009).

4.2.5. Proposed alternatives to the F-measure

Some alternative measures have been proposed as having superior properties over F. I present here two of these proposed alternatives to the F-measure: a modification of the *slot error rate* (Makhoul et al., 1999) and the R-value (Räsänen et al., 2009).

A standard method to assess the performance of an ASR system is to measure the *word error rate* (WER). With respect to speech segmentation, this qualifies as an indirect measure. It is defined as the edit distance between a hypothesised sequence of words and the reference sequence of words, counting *insertions*, *deletions* and *substitutions*:

$$WER = \frac{insertions + deletions + substitutions}{words\,in\,reference}$$

Makhoul et al. (1999, p.249–252) propose to apply this kind of measure in general for information extraction from discrete data. They call this measure the *slot error rate*. Insertions and deletions can be identified with false positives and false negatives, respectively. Substitutions, however, do not fit into the terminology set up so far for this section. They are defined as assignments of a wrong label at an otherwise correct location. Omitting the number of substitutions, a measure can be defined which relates the total number of boundary errors (insertions and deletions) to the total number of reference boundaries:

$$S' = \frac{FN + FP}{|ref|} \tag{4.5}$$

If the above equation is interpreted according to the definition of the slot error rate, the denominator must be replaced by the total number of available slots which is the number of frames of the digitised signal, i.e. the length of the signal in seconds times

the sampling rate[11]. However, using the total number of reference boundaries $|ref|$ instead, makes the measure easier to interpret: values < 1.0 indicate cases where the total number of boundary errors is less then the expected number of boundaries. The results would be equivalent if the total number of frames were used as denominator in equation 4.5, but the magnitude of the numerical values would be much smaller[12].

The R-value is supposed to be more sensitive to over-segmentation (OS) errors than F. It has a maximum value of 1 at a "target point" in an abstract OS × HR space (defined as 100% HR and 0% OS). The measure is defined as:

$$r_1 = \sqrt{(100 - \mathrm{HR})^2 + (\mathrm{OS})^2} \tag{4.6}$$

$$r_2 = \frac{-\mathrm{OS} + \mathrm{HR} - 100}{\sqrt{2}} \tag{4.7}$$

$$R = 1 - \frac{|r_1| + |r_2|}{200} \tag{4.8}$$

4.2.6. Pk and WindowDiff

A measure originally proposed for text segmentation is *WindowDiff* (WD) (Pevzner and Hearst, 2002). It is based on the P_k measure (Beeferman et al., 1999), and its basic idea is to consider *near misses* and to prevent "cheating" from algorithms that attempt to game the evaluation measure (e.g., achieving higher recall or precision by heavily over- or under-segmenting).

The two measures use a sliding window over the data sequence to compare the hypothesised segmentation *hyp* and the reference *ref*, shifting the window iteratively by one frame. While P_k counts the instances where the two segmentations disagree at the window end-points, *WindowDiff* counts the instances where the number of boundaries within each window disagrees. I follow Pevzner and Hearst (2002), defining P_k and *WindowDiff* as follows[13]:

$$P_k(ref, hyp) = \frac{1}{N-1} \sum_{i=1}^{N-k} f(ref, hyp, i, i+k) \tag{4.9}$$

[11]The terms *frame* and *sample* are considered synonymous in the context of digital audio recordings in this thesis.

[12]Note that the total number of reference boundaries $|ref|$ is never greater than the total number of frames of the signal, if a suitable sampling rate is used.

[13]For consistency, the two definitions from Pevzner and Hearst (2002) are used according to which both measures give a score of 0 for perfect segmentations. P_k is originally defined differently by Beeferman et al. (1999), resulting in a score of 1 for a perfect segmentation.

with:

$$f(s_1, s_2, i, j) = \begin{cases} 1 & \text{if} \quad (\#b(s_1, i, j) > 0) \\ & \quad\quad\quad \neq \\ & \quad\quad\quad (\#b(s_2, i, j) > 0) \\ 0 & \text{else} \end{cases} \tag{4.10}$$

and:

$$\begin{aligned} &WindowDiff(ref, hyp) = \\ &\frac{1}{N-k} \sum_{i=1}^{N-k} (|\#b(ref, i, i+k) - \#b(hyp, i, i+k)| > 0) \end{aligned} \tag{4.11}$$

where $\#b(x, i, j)$ is the number of boundaries in sequence x between index i and j; k is the size of the window and N is the total number of primitive units in the data (e.g. sentences or words in text segmentation or audio frames or samples as in our present case of acoustic speech segmentation). The window size k is set to half the mean segment size in *ref* for both P_k and *WindowDiff*.

4.2.7. Relative TP

In addition to the well known evaluation measures listed so far, I propose one which has not been commonly used so far. It is similar to a measure proposed by Ziółko et al. (2007), which they call "fuzzy recall and precision". They base their evaluation method on fuzzy set theory and redefine precision and recall accordingly. The set of all hypothesised boundaries is defined as a fuzzy set with a membership function $f(x) \in [0, 1]$. They assume a reference segmentation which defines "ranges" in which the segments overlap (p.3). The hypothesised segmentation, on the other hand, is assumed to be non-overlapping. If a hypothesised boundary h falls into this overlapping region, it is counted as correct with a membership degree of $f(h) = 1$. Otherwise, the membership degree is defined by the distance from the reference boundary in relation to the segment length. Unfortunately, the URL for the source code published by Ziółko et al. (2007, p.4) was not available at the time this section was written.

I define a quantity called *relative TP*, for which the symbol $\widetilde{TP}$ is used in the following text. Assuming non-overlapping segmentations as before, relative TP is based on an alignment of a hypothesised segmentation *hyp* and a reference segmentation *ref*. The alignment is defined in the same manner as in section 4.2.2 with the exception that the third condition is not applied — i.e. two boundaries are aligned without reference to a pre-defined tolerance distance Δt. Thus, the adjusted alignment definition reads as follows:

Definition 4.2 *A hypothesised boundary $h \in hyp$ is assigned to a reference boundary $r \in ref$ iff:*

- *h is the closest boundary in hyp to r and*
- *r is the closest boundary in ref to h*

Let the set of all pairs of segment boundaries (h_i, r_i) aligned according to definition 4.2 be denoted by the symbol $\mathcal{A}$. Further, let ς_i be half the length of the segment into which a hypothesised boundary h_i falls, with

$$\varsigma_i = \begin{cases} \frac{r_i - r_{i-1}}{2} & \text{if} \quad h_i < r_i \\ \frac{r_{i+1} - r_i}{2} & \text{else} \quad . \end{cases}$$

If $d_i = |r_i - h_i|$ is the distance between a boundary h_i and its matching boundary r_i, the weight $w(h_i)$ of boundary h_i is defined as

$$w(h_i) = \frac{\varsigma_i - d_i}{\varsigma_i} \quad .$$

The quantity $\widetilde{TP}$ is defined as the weighted sum of all aligned boundary pairs $(h_i, r_i) \in \mathcal{A}_{\text{hyp,ref}}$, with:

$$\widetilde{TP}(ref, hyp) = \sum_{i=1}^{|\mathcal{A}|} w(h_i) = \sum_{i=1}^{|\mathcal{A}|} \frac{\varsigma_i - d_i}{\varsigma_i} \quad . \tag{4.12}$$

Figure 4.3 on page 109 shows an illustration of the principle behind $\widetilde{TP}$. In this figure, c_i marks the centre of the segment which begins at time r_{i-1} and ends at r_i, and c_{i+1} marks the centre of the segment which begins at time r_i and ends at r_{i+1}, with

$$c_i = r_i - \frac{r_i - r_{i-1}}{2} \quad \text{and} \quad c_{i+1} = r_i + \frac{r_{i+1} - r_i}{2} \quad .$$

Two possible locations of a hypothesised boundary that might be aligned with r_i are marked by a and b. A case where the hypothesised boundary precedes its matching reference boundary is illustrated by a, and a case where the reference boundary precedes the hypothesised boundary is illustrated by b, respectively. Note that the distance between a and r_i is equal to the distance between b and r_i. However, the corresponding weight $w(a)$ is smaller than $w(b)$, since b is relatively closer to r_i. The weight of a hypothesised boundary with exactly the same time index as r_i is 1, while a hypothesised boundary exactly at the centre of a segment is 0. The centre of the corresponding segment is, by definition, the maximum distance for any hypothesised boundary from its corresponding reference.

With this definition of $\widetilde{TP}$ it is possible to define measures of *relative precision* $\widetilde{\text{prc}}$ and *relative recall* $\widetilde{\text{rec}}$ analogous to the definitions in 4.2 and 4.3 as follows:

$$\widetilde{\text{prc}} = \frac{\widetilde{TP}}{|hyp|} \tag{4.13}$$

$$\widetilde{\text{rec}} = \frac{\widetilde{TP}}{|ref|} \tag{4.14}$$

Based on these measures, a *relative F-measure* can be defined as:

$$\widetilde{F}_\beta = \frac{(\beta^2+1)\,\widetilde{\text{prc}} \cdot \widetilde{\text{rec}}}{\beta^2\,\widetilde{\text{prc}} + \widetilde{\text{rec}}} \tag{4.15}$$

Since precision and recall are both based on the number of true positives TP, it is sufficient to adapt the definition of TP in order to modify these measures. The total number of true positives depends on an arbitrary distinction between correct and wrong predicted boundaries. The alignment definition 4.1 is based on a tolerance window to distinguish between matches on the one hand and insertions or deletions on the other hand. It does not consider how good a match is. All matched hypothesised boundaries get a weight of 1. Replacing this alignment definition by one which does not refer to a fixed tolerance around a reference boundary, and assigning a weight to each matching boundary pair according to the maximum distance, a measure is obtained which is sensitive to displacements of boundaries relative to the reference segment length.

The weight is defined here to decrease from 1 to 0 according to the distance of a hypothesised boundary to its matching reference boundary in relation to the centre of the corresponding segment. However, the weight function could take other forms, for example having a bell-shaped curve around the reference boundary. For the sake of simplicity, I use the linear weighting.

4.2.8. Evaluation measures for speech segmentation: Summary

There is a wide range of different evaluation measures which have been proposed and used to assess the quality of speech segmentations. The list of measures discussed in this section is not exhaustive. Measures which take into account the labels associated with a segment have been omitted. The main focus in this section is on speech segmentation as a task which is distinct from the identification of speech segments. Therefore, only the quality of the segment boundaries needs to be assessed. The slot error rate, for example, has been explicitly modified as shown in equation 4.5 to exclude the labels.

All evaluation measures discussed in this section are direct, external measures according to the terminology defined in section 3.6. In contrast to the evaluation of

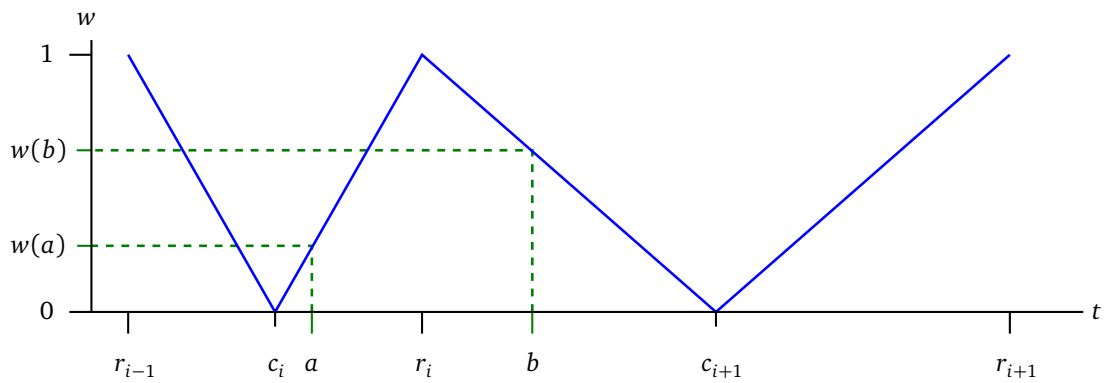


Figure 4.3.: Illustration of the "relative TP" boundary weight curve around a boundary at location r_i: a hypothesised boundary which is located exactly at r_i gets a weight of 1. The closer a boundary is to the centre of the segment ending or starting at r_i, the closer to 0 the weight of that boundary is.

data clustering solutions (cf. section 5.2.4 on page 168), there are, to the best of my knowledge, no widely used internal evaluation measures in the domain of speech segmentation[14]. As mentioned in the previous section, there is relatively little work on unsupervised speech segmentation. In case of supervised speech segmentation, there is no need for internal evaluation measures, since the result can be compared against the given reference. Miller et al. (2009, p.4f) address the evaluation of speech segmentations without reference to an a priori given ground truth. They resort to manual evaluation of their data, however. One example of an indirect measure was mentioned in this section: the word error rate. It is measured on the result of an ASR system, and can be used as an indirect indicator for the quality of the underlying speech segmentation based on the assumption that better segmentations result in lower word error rates. Another example of an indirect evaluation of speech segmentation is the work by Sharma and Mammone (1996a, p.96). They evaluate their proposed "blind" speech segmentation method by measuring the performance of a speaker verification system.

Mean deviation or distance is sometimes used to asses the quality of a segmentation. This is a valid measure to detect a constant bias of a segmenter. If a segmenter has a relative bias (for example one which consistently places boundaries at the centre of the segments or one, which (more or less) consistently displaces boundaries by 10% of the true segment length, etc.) the mean deviation will be less informative of that fact. As for any reported mean value, the corresponding variance or standard deviation statistic should be reported.

[14]Note that *log-likelihood* estimates, for example, are often employed in ASR in the learning process to create optimal models of the observed training data, i.e., to find the model with maximum likelihood (cf. Sharma and Mammone, 1996b). Although such *objective functions* could be applied as *internal evaluation measures*, they are not considered as such in this thesis.

Comparing the evaluation measures described in this section, some basic properties can be identified. Some evaluation measures assume a distinction between correct and wrong boundaries in the hypothesised solution, and they are defined based on TP (and the related numbers FP and FN). Such measures do not directly take into account local properties of a segmentation, like how far a given hypothesised boundary is from its true reference boundary location. Some evaluation measures, assume the segmented data to be composed of a sequence of primitive (or fundamental) units. Such measures are based on the assumption that only a fixed number of boundaries is possible at any given part of the sequence and thus, that there is a maximum number of possible boundary locations. As mentioned earlier, this distinction is only a theoretical one, since in practice, any digital speech signal is a discrete representation. However, this discreteness of a digital speech signal depends only on technical limitations and is not formally required. The following section presents an empirical analysis of the properties of the evaluation measures discussed in this section.

4.3. Experiment: Comparing evaluation measures

This section presents a detailed experimental study on the properties of external segmentation quality measures. The source code for these experiments is listed in appendix B.1.1 on page 261.

In order to compare the various evaluation measures listed above and to examine systematic weaknesses or strengths of each evaluation measure under consideration, I implemented a series of diagnostic segmentation methods. The evaluation measures considered in this chapter are based on a comparison of true, gold-standard boundaries and hypothesised boundaries. They require binary decisions, i.e. distinctions between correct and wrong hypotheses. Thus, the employed diagnostic segmentation methods make only binary decisions for the sake of simplicity (instead of, for example, implementing a boundariness function on the input data producing continuous degrees of boundariness at each time index). I adopt the common assumption that a manually created reference segmentation provides a useful definition of valid segment boundaries and that hypothesised automatic segmentations can be evaluated against such a gold standard[15].

4.3.1. Test data

The definitions of the diagnostic segmenters for this study are based on an existing reference segmentation. For the current experiment, the reference boundaries are taken from the following three speech corpora:

[15]See the discussion in section 4.2.2.

- The IMS unit selection corpus — German (cf. appendix A.2)

  Used subsets:

  - Male speaker: labelled "IMS:ms"
  - Female speaker: labelled "IMS:rk"

- The Polish BOSS corpus (cf. appendix A.3)

  Used subsets:

  - Male speaker, part one: labelled "BOSS:A"
  - Male speaker, part two: labelled "BOSS:B"

- The MOCHA-TIMIT corpus — English (cf. appendix A.4)

  Used subsets:

  - Female speaker: labelled "MOCHA:fsew"
  - Male speaker: labelled "MOCHA:msak"

The original sampling frequency of the audio data for all corpora used in this experiment is 16 kHz. The effective time resolution has been changed to 500 frames per second for computational efficiency for the experiments presented later in this chapter (section 4.4). The same data is used here as the basis for the diagnostic segmentation methods. A frame size of 2 ms is a reasonable resolution for the purpose of the present study. By applying the diagnostic segmenters to a reference which is derived from a real speech data set instead of creating a completely artificial one, the evaluation measures are confronted with realistic problems. Yet, the properties of the evaluated data are well known due to the fully controlled properties of the diagnostic segmentations. The statistics for the evaluated data sets are listed in table 4.2. The corpora cover three languages (German, Polish and English) for both female and male speakers.

4.3.2. Method: Diagnostic segmenters

A variety of diagnostic segmenters is defined in order to generate a number of different segmentations of the test data with known properties.[16] The following segmentations are defined in reference to a given true segmentation such that the hypothesised boundaries are generated in relation to the true segment boundaries. As mentioned in the previous section, these segmentation methods define binary boundariness functions, giving a

[16] The approach of using artificial data sets and applying evaluation measures to them is often taken to investigate the properties of different evaluation measures and to assess their applicability to certain problems. The study by Gurrutxaga et al. (2011) is an example in the domain of cluster analysis (see section 5.2 on page 159).

data set	total number of segments	segment length (s) mean	sd	data covered by tolerance windows $\Delta t = 10\,\text{ms}$ % frames	segments	$\Delta t = 20\,\text{ms}$ % frames	segments
IMS:ms	110,443	0.1859	0.7467	10.76	367	21.12	11,976
IMS:rk	90,936	0.1177	0.2816	16.99	357	33.49	7,528
MOCHA:fsew	15,243	0.1204	0.1765	16.60	0	32.42	1,812
MOCHA:msak	15,243	0.1188	0.2042	16.83	0	32.59	2,403
PL-BOSS:A	11,639	0.0711	0.0362	28.13	19	55.59	996
PL-BOSS:B	2,292	0.0832	0.0324	23.98	7	47.77	69

Table 4.2.: The segment statistics for the data sets used with the diagnostic speech segmentations. Columns five and seven show the percentage of all data frames which are covered by a tolerance window for two evaluated tolerances of $\Delta t \in \{10, 20\}$ ms, respectively, and columns six and eight show the number of segments which are totally covered by the tolerance windows around their boundaries.

value of 1 at a hypothesised boundary location and 0 else over the total range of time indices of the corpora. A unique symbol is associated with each diagnostic segmentation method, as follows:

- seg_{ref}: this segmenter generates a perfect segmentation with $hyp(seg_{\text{ref}}) = ref$. This is a trivial diagnostic segmenter which assigns the true reference boundaries to the input data. This represents the upper bound for real segmentation methods.

- seg_{all}: this segmenter generates a total segmentation of the speech stream with a boundary at every frame. Since any digital speech signal is composed of a finite sequence of time-discrete frames (samples) determined by the sampling rate of the signal, there is a maximum number of boundaries that can be assigned to the input data. This segmenter produces such a maximum segmentation of the input.

- $seg_{\Delta t+1}$: this segmenter generates boundaries which are shifted by $\Delta t + 1$ frame relative to the reference boundaries in ref. The parameter Δt is equal to the corresponding tolerance window parameter used for evaluation. As a consequence, each boundary produced by this segmenter is just outside of the tolerance window surrounding the corresponding reference boundary.

- $seg_{\Delta t}$ and $seg_{\Delta t-1}$: these segmenters shift each boundary by Δt and $\Delta t - 1$ frame, respectively, relative to the reference segmentation.

- seg_{cent}: this segmenter generates a segmentation which places a boundary at the centre of each true segment. As far as the distance from a hypothesised boundary

to the nearest reference boundary is concerned, this segmenter produces the worst possible segmentation that can be achieved without over- or under-segmenting the input data.

- seg_{double}: this segmenter generates two boundaries for each reference boundary which are placed ± 1 frame from its true location.

  This segmenter doubles the number of segment boundaries and places all of them at a wrong location which is minimally away from the true location. An evaluation measure which counts every boundary within a specific tolerance window as correct must recognise that there are actually two candidate boundaries which are equally close to the reference location. Counting both as correct will ignore the fact that the total number of boundaries is twice the true number.

- seg_{shift}: this segmenter assigns boundaries to the input data which are shifted by 1 frame relative to the corresponding true boundaries in *ref*. A shift of one frame is the smallest possible deviation for any given sampling frequency of the data.

- seg_{allC}: this segmenter cuts every reference segment in half. All original boundaries are copied from *ref*, and in addition to these, a new boundary is introduced at the temporal midpoint of each segment. This doubles the number of segments.

- seg_{half}: this segmenter drops every second reference boundary. This cuts the total number of segments by half relative to the true number.

- seg_{ten}: this segmenter shifts each reference boundary by one tenth of the original segment length.

  This segmenter is defined such that only internal reference boundaries are considered by the algorithm. This avoids the introduction of a false positive boundary at the very beginning of the corpus, but this also changes the length of the first segment, while the last segment will be shorter than its corresponding reference.

Two baseline segmentation methods are also defined and evaluated in addition to the diagnostic segmenters defined above. As discussed in section 3.6, applications in speech and natural language processing or machine learning are commonly compared against a baseline method that often represents the simplest possible solution to the problem at hand, usually involving (pseudo-) random decisions.

- seg_{const}: this segmenter creates a segmentation with $\alpha \cdot |ref|$ equally sized segments (with one potentially shorter segment if the corpus length cannot be divided by the required number of segments without a remainder). This is a simple baseline segmenter which is based on a single parameter: the total number of desired segments. If the sequence of labels is assumed to be known (as, for example, in

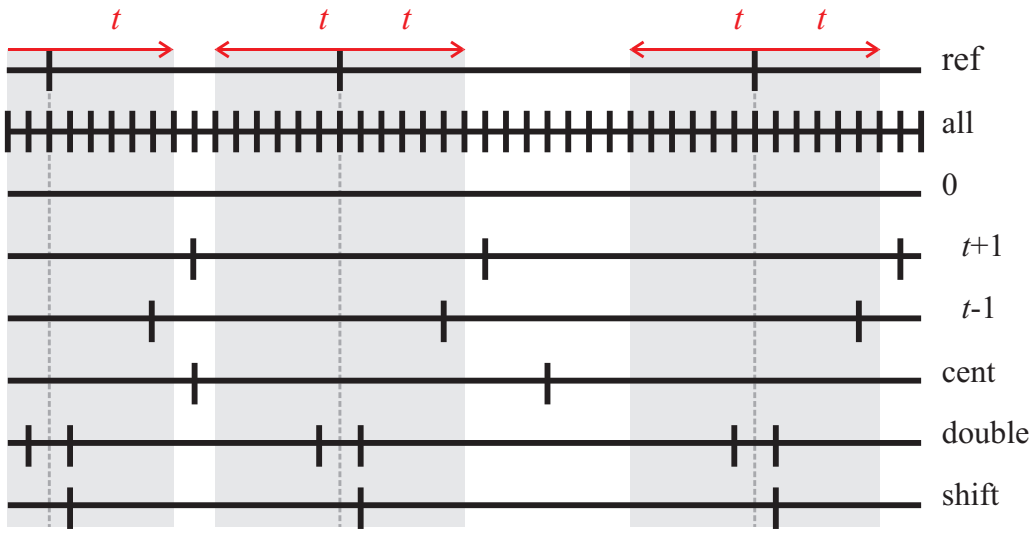


Figure 4.4.: Schematic illustration of diagnostic segmenters: Horizontal lines represent the time axis of a data sequence with segment boundaries shown as short vertical lines. The vertical, grey dashed lines indicate the true boundary location. The top row shows $seg_{\text{ref}} = ref$, the second row shows seg_{all}, etc. The grey background area depicts the tolerance window.

some ASR applications), this constitutes a realistic baseline against which every segmentation method has to be compared.

- seg_{rand}: this segmenter creates a segmentation with $\alpha \cdot |ref|$ segments whose lengths are taken from a normal distribution with parameters μ and σ estimated from the reference distribution (see table 4.2 on page 112). This baseline segmenter is slightly more complex than the previous one as it incorporates more a priori knowledge about the data. However, the assumed normal distribution of segment lengths will not correspond to the real distribution of segment lengths in the reference corpus. Two additional parameters l_{min} and l_{max} are defined for the actual implementation to limit the generated segment lengths. This is necessary since values drawn from a normal distribution could be 0 or negative or lead to unrealistically long segments. I set l_{min} to the real minimum length of the reference segments and $l_{\text{max}} = \mu + 2\sigma$. Both length parameters are scaled according to α.

Figure 4.4 on page 114 shows an illustration of some of these diagnostic segmenters. I set the parameter α to 0.5, 1.0 and 2.0, and Δt to 10 and 20 ms for the present experiments.

The absolute temporal distance between the boundaries in $seg_{\Delta t+1}$ and $seg_{\Delta t}$ is comparable with the distance between the boundaries in seg_{shift} and seg_{ref}. Intuitively, a

quality measure should reflect this fact by penalising seg_{shift} in relation to seg_{ref} by the same magnitude as it penalises $seg_{\Delta t+1}$ in comparison to $seg_{\Delta t}$. This should hold true for measures which are metrics.

The segmenters $seg_{\Delta t+1}$, $seg_{\Delta t}$, $seg_{\Delta t-1}$ and seg_{shift} simulate systematic differences between the generated segmentation and the reference. Baghai-Ravary et al. (2009, p.2879) point out that "such consistent discrepancies do not indicate a significant difference in precision between alignment systems" and they should not actually be counted as errors. Ljolje et al. (1997, p.309f) compute the bias (i.e. the mean difference in boundary placement between the system and the reference) of their automatic segmentation algorithm for boundaries between different phonetic categories (e.g. 'unvoiced stop to vowel', 'vowel to vowel', etc.). They explicitly report the segmentation results before and after the removal of the system's bias.

The difference between the two segmenters $seg_{\Delta t-1}$ and $seg_{\Delta t+1}$ is important for all measures which depend on the definition of a tolerance window around each reference boundary.

The segmenters seg_{cent} and seg_{ten} introduce a constant error with respect to the true segment lengths by placing a boundary consistently at the centre of each segment or at 10% of its length.

4.3.3. Implementation

The diagnostic segmenters and the various evaluation measures for this experiment are implemented in JAVA. The commented source code is listed in appendix B.1.1 starting on page 261. The experiment is started with the `main`-method in the class `RunSegmentationDiagnostic` (cf. listing B.1). This method reads the program parameters, initialises the corpus data and initiates the segmentation evaluations. An example call of the program with all required arguments using `java` from the command line looks as follows:

```
java -Xmx10G -Xms1G -cp bin/:. de.uniStuttgart.danielDuran.↩
   RunSegmentationDiagnostic IMSms.500.frames 500 UTF-8 OUTPUTDIRECTORY 10.0,20.0 ↩
   1.0,0.5,2.0 1000
```

where "`IMSms.500.frames`" is the name of the input file, followed by the corresponding frame rate in Hertz, the input file encoding (UTF-8), the output directory, a comma-separated list of tolerance sizes Δt in ms, a comma-separated list of α values for the baseline segmenters and the number of repetitions of the random baseline segmenter[17].

[17] The first parameters control the `java` tool which starts a Java virtual machine running the Java program: "`-Xmx10G`" specifies the maximum size of working memory allocated by the virtual machine for the application to be 10 gigabytes, "`-Xms1G`" specifies the initial size of working memory reserved for the program to be 1 gigabyte, and "`-cp ...`" adds the specified directories and files to Java's class path. Note that `Xmx` is higher than the actually required memory. Java applications tend to run faster if the

The input is read from a plain text file where the label is given for each data frame on a separate line (implemented in class `CorpusReader`, cf. listing B.2). For each tolerance window size, the methods of the class `Comparison` (listing B.6) are called to run the diagnostic segmenters on the input data. The class `Evaluator` (listing B.21) takes a generated input segmentation and the reference from the corpus, does the alignment and counts the matches and all other necessary statistics and stores the evaluation results in an object of class `EvaluationResults` (listing B.22). The methods print out statistics about the input data and the values of the various evaluation measures. In addition to the information printed to standard out, text files are produced for each run with the results of the evaluation measures, formatted in both LaTeX-tabular format and CSV, respectively, for printing and further analysis. See commented source code in listings B.1–B.23 for details.

4.3.4. Results and discussion of diagnostic segmentations

The results for the various evaluation measures on the diagnostic segmentations with a tolerance window with $\Delta t = 10\,\text{ms}$ are shown in tables 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8 on pages 119–124. The results for a tolerance window with $\Delta t = 20\,\text{ms}$ are shown in tables 4.9, 4.10, 4.11, 4.12, 4.13 and 4.14 on pages 125–130.

The columns show the results obtained by the different evaluation measures. Additionally, the tables also show the total number of boundaries in *hyp* that could be aligned with boundaries in *ref* in column "aligned". The corresponding total number of true positives is shown in column "*TP*" (as determined by definition 4.1 given in section 4.2.2). Of the measures derived from precision (column prc) and recall (column rec), only the results for the F_1-measure are shown. The results for HR, FA and MR are not shown separately because they can be trivially derived from precision and recall. The quantity defined above as relative TP is shown in column "$\widetilde{TP}$", along with the corresponding values for relative precision, relative recall and relative F_1-measure in columns "$\widetilde{\text{prc}}$", "$\widetilde{\text{rec}}$" and "$\widetilde{F}_1$", respectively.

The rows of the tables correspond to the various segmentations of the data sets generated by the diagnostic segmenters. The upper part of the table shows the results for the diagnostic segmenters which do not depend on the global parameter α. The lower part shows the results for the three parameter settings of $\alpha \in \{\frac{1}{2}, 1, 2\}$, which determines the total number of segments generated by seg_{const} and seg_{rand}. The results for seg_{rand} are averaged over 1000 repeated runs with independently generated random segmentations.

The true number of segments is shown in table 4.2 on page 112 for each data set. It can also be inferred from the evaluation results tables 4.3–4.14: the number of aligned

actually used memory is not close to its reserved capacity. The experiments were carried out on a Linux system and took several minutes in total to complete for all input corpora.

boundaries for seg_{ref} corresponds to the true number of segments.

A comparison between the columns "aligned" and "TP" shows that the number of hypothesised boundaries which can be aligned with the reference can be significantly lower than the number of hypothesised boundaries which are actually counted as correct because of the additional condition in the definition of the alignment procedure that the hypothesised boundary must be within a set tolerance distance from the reference boundary (cf. section 4.2.2). Thus, all measures which depend on the so defined number of true positives TP show different results depending on the parameter Δt. Compare, for example, prc, rec, F_1 and S' in tables 4.3 and 4.9: the scores in the latter table tend to be higher in comparison to the scores in the former table because of the higher tolerance[18]. A notable exception is visible in $seg_{\Delta t}$, $seg_{\Delta t+1}$ and $seg_{\Delta t-1}$ which are defined such that they displace the boundaries by about the size of the tolerance window. In doing so, the total number of aligned boundaries decreases if the shift is increased from approximately 10 ms to approximately 20 ms. Due to the larger tolerance window, more boundaries get misaligned[19]. This shift also affects P_k, WD, $\widetilde{\text{prc}}$, $\widetilde{\text{rec}}$ and $\widetilde{F}_1$, which actually do not depend on a fixed tolerance window parameter. Note that except for the segmentations produced by $seg_{\Delta t}$, $seg_{\Delta t+1}$ and $seg_{\Delta t-1}$, these five evaluation measures give identical results for $\Delta t = 10$ ms and $\Delta t = 20$ ms. The values for seg_{rand} are slightly different because of the random placement of boundaries which is also visible in slight differences in the number of aligned boundaries. The effect of the tolerance window is clearly visible for TP in tables 4.3 and 4.9: despite the relatively small differences in the number of aligned boundaries for seg_{rand} (11,134 vs. 11,160 for $\alpha = 0.5$; 19,561 vs. 19,605 for $\alpha = 1$; and 34,529 vs. 34,468 for $\alpha = 2$), there is a large difference in the number of true positives TP (2,509 vs. 4,654 for $\alpha = 0.5$; 5,075 vs. 9,467 for $\alpha = 1$; and 10,178 vs. 19,043 for $\alpha = 2$). Thus, by simply increasing the tolerance window size, the apparent performance of a segmenter can be significantly improved according to measures which rely on the common definition of TP.

Precision and recall are widely used, but neither takes into account all error types, and they can each be improved at the cost of the other. Recall can be increased by inserting more boundaries, which, in turn, decreases precision. A comparison of the results of the segmentations of seg_{const} and seg_{rand} in tables 4.3–4.14 shows that recall increases as more boundaries are inserted. This way the F_1-measure can be improved

[18] Note that I use the expression "higher score" to refer to a value given by an evaluation measure which is indicative of a higher segmentation quality, independent of the fact whether a particular measure assigns higher numeric values to better segmentations (like F_1), or whether it assigns lower numeric values to better segmentations (like S').

[19] For the IMS:ms data set, for example, the number of overlapping tolerance windows increases from 194 with a tolerance of ±10 ms to a total of 10,129 overlapping tolerance windows with a tolerance of ±20 ms. The number of segments which are completely covered by tolerance windows around their boundaries increases from 367 to 11,967. For the PL-BOSS:A data set, the number of overlapping tolerance windows increases accordingly from 12 to 796, and the number of totally covered segments from 19 to 996. Similar effects can be observed on the other data sets.

without any essential improvement of the employed segmenter, thus giving a false impression of the segmenter's performance — although F_1 is supposed to compensate for over-segmentation by taking into account both precision and recall.

By definition, OS does not consider the quality of the individual segments but only the total number of boundaries. The segmentation generated by seg_{cent}, for example, always gets a perfect OS score just because of the correct total number of boundaries, although it is essentially a useless segmentation. Note that the additional boundary introduced by seg_{cent} is not visible in the results shown in tables 4.3–4.14 due to rounding. The actual value for the data set IMS:ms, for example, is around OS = 0.0009%. However, in cases where such a high degree of precision is required, it is more appropriate to directly compare the actual numbers of segments (i.e. 110,444 in seg_{cent} vs. 110,443 in the reference in the case of data set IMS:ms). A perfect score is assigned to all segmentations which are defined as shifts of the true boundaries (i.e. $seg_{\Delta t+1}$, $seg_{\Delta t}$, $seg_{\Delta t-1}$, seg_{shift} and seg_{ten}). Therefore, OS should not be used without an appropriate measure of the boundary locations' accuracy. If an evaluation measure is required to distinguish between a perfect solution and a minimally worse segmentation, the precision of the computations and representation of the results must be taken into account (as the example of over-segmentation shows where only 1 false boundary is added to a sequence with more than 100,000 boundaries in total).

Test criteria

The results in tables 4.3–4.14 allow a systematic investigation of the desired properties of evaluation measures as defined above. In order to carry out a systematic evaluation of these results, I define a number of simple scoring functions which assign a value of 1 to an evaluation result which is in agreement to an expectation based on the assumption that the evaluation measure has a particular property. One such test criterion is, for example, whether seg_{ref} gets the highest evaluation score. This means that the perfect solution is assigned the highest score by an evaluation measure. If this condition is true for a given evaluation measure, the scoring function returns 1 (assigning one score point to this result) and otherwise 0. These scoring functions are formulated as a set of test criteria and they are applied to the evaluation results obtained by a given evaluation measure on one given data set at a time.

Let $s(hyp, ref) \in \mathbb{R}$ be an external evaluation measure assigning a score to a given segmentation hyp in reference to another solution ref. For the sake of notational simplicity, s is interpreted in the following definitions as assigning higher numbers (scores) to better solutions.

The test conditions are defined informally as follows:

(A) Test whether $s(hyp(seg_{\text{ref}}), ref) > s(hyp', ref)$, for all other segmentations hyp' on the same data with the same tolerance parameter setting. This is true if the

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	110,443	110,443	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	110,443.00	1.00	1.00	1.00
seg_{all}	110,443	110,443	0.01	1.00	0.02	-77.47	9,193.71	91.94	0.64	1.00	110,443.00	0.01	1.00	0.02
seg_{half}	55,222	55,222	1.00	0.50	0.67	0.65	-50.00	0.50	0.12	0.24	55,222.00	1.00	0.50	0.67
seg_{allC}	110,443	110,443	0.50	1.00	0.67	0.15	100.00	1.00	0.05	0.38	110,443.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	108,813	1,118	0.01	0.01	0.01	0.16	0.00	1.98	0.03	0.11	67,408.11	0.61	0.61	0.61
$seg_{\Delta t}$	109,795	109,795	0.99	0.99	0.99	0.99	0.00	0.01	0.02	0.09	74,616.78	0.68	0.68	0.68
$seg_{\Delta t-1}$	110,251	110,251	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.08	81,835.85	0.74	0.74	0.74
seg_{cent}	56,452	1,094	0.01	0.01	0.01	0.15	0.00	1.98	0.09	0.23	1,256.50	0.01	0.01	0.01
seg_{double}	110,443	110,443	0.50	1.00	0.67	0.15	100.00	1.00	0.01	0.36	103,309.14	0.47	0.94	0.62
seg_{shift}	110,442	110,442	1.00	1.00	1.00	1.00	0.00	0.00	0.01	0.02	103,308.23	0.94	0.94	0.94
seg_{ten}	107,657	92,010	0.83	0.83	0.83	0.86	0.00	0.33	0.05	0.10	85,656.25	0.78	0.78	0.78
$\alpha = 0.5$														
seg_{const}	24,489	6,535	0.12	0.06	0.08	0.31	-50.00	1.38	0.43	0.46	13,552.96	0.25	0.12	0.16
seg_{rand}	11,134	2,509	0.05	0.02	0.03	0.28	-50.00	1.45	0.39	0.40	6,383.64	0.12	0.06	0.08
$\alpha = 1$														
seg_{const}	45,738	13,113	0.12	0.12	0.12	0.25	0.00	1.76	0.50	0.56	24,446.85	0.22	0.22	0.22
seg_{rand}	19,561	5,075	0.05	0.05	0.05	0.19	0.00	1.91	0.42	0.44	10,901.20	0.10	0.10	0.10
$\alpha = 2$														
seg_{const}	79,945	26,257	0.12	0.24	0.16	-0.25	100.00	2.52	0.63	0.76	44,275.25	0.20	0.40	0.27
seg_{rand}	34,529	10,178	0.05	0.09	0.06	-0.35	100.00	2.82	0.47	0.51	19,163.81	0.09	0.17	0.12

Table 4.3.: Evaluation results for diagnostic segmentations on dataset “IMS:ms” with $\Delta t = 10\,\text{ms}$. For details see page 116.

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	90,936	90,936	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	90,936.00	1.00	1.00	1.00
seg_{all}	90,936	90,936	0.02	1.00	0.03	-48.36	5,782.72	57.83	0.54	1.00	90,936.00	0.02	1.00	0.03
seg_{half}	45,468	45,468	1.00	0.50	0.67	0.65	-50.00	0.50	0.21	0.25	45,468.00	1.00	0.50	0.67
seg_{allC}	90,936	90,936	0.50	1.00	0.67	0.15	100.00	1.00	0.20	0.48	90,936.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	89,584	952	0.01	0.01	0.01	0.16	0.00	1.98	0.13	0.17	58,345.74	0.64	0.64	0.64
$seg_{\Delta t}$	90,327	90,327	0.99	0.99	0.99	0.99	0.00	0.01	0.11	0.15	63,802.22	0.70	0.70	0.70
$seg_{\Delta t-1}$	90,747	90,747	1.00	1.00	1.00	1.00	0.00	0.00	0.09	0.12	69,310.65	0.76	0.76	0.76
seg_{cent}	46,363	950	0.01	0.01	0.01	0.16	0.00	1.98	0.38	0.43	928.33	0.01	0.01	0.01
seg_{double}	90,934	90,934	0.50	1.00	0.67	0.15	99.99	1.00	0.02	0.46	85,550.00	0.47	0.94	0.63
seg_{shift}	90,930	90,930	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	85,544.43	0.94	0.94	0.94
seg_{ten}	89,891	70,390	0.77	0.77	0.77	0.81	0.00	0.45	0.13	0.15	71,624.04	0.79	0.79	0.79
$\alpha = 0.5$														
seg_{const}	33,390	8,534	0.19	0.09	0.13	0.34	-50.00	1.31	0.48	0.49	17,588.72	0.39	0.19	0.26
seg_{rand}	18,219	4,416	0.10	0.05	0.06	0.30	-50.00	1.40	0.47	0.47	9,675.52	0.21	0.11	0.14
$\alpha = 1$														
seg_{const}	60,015	16,978	0.19	0.19	0.19	0.31	0.00	1.63	0.50	0.52	32,611.90	0.36	0.36	0.36
seg_{rand}	33,307	8,868	0.10	0.10	0.10	0.23	0.00	1.80	0.48	0.49	17,810.53	0.20	0.20	0.20
$\alpha = 2$														
seg_{const}	83,585	33,756	0.19	0.37	0.25	-0.17	100.00	2.26	0.54	0.57	53,463.85	0.29	0.59	0.39
seg_{rand}	53,365	17,423	0.10	0.19	0.13	-0.28	100.00	2.62	0.50	0.52	30,925.02	0.17	0.34	0.23

Table 4.4.: Results for dataset “IMS:rk”. $\Delta t = 10\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	15,243.00	1.00	1.00	1.00
seg_{all}	15,243	15,243	0.02	1.00	0.03	-49.55	5,922.50	59.22	0.55	1.00	15,243.00	0.02	1.00	0.03
seg_{half}	7,622	7,622	1.00	0.50	0.67	0.65	-50.00	0.50	0.20	0.25	7,622.00	1.00	0.50	0.67
seg_{allC}	15,243	15,243	0.50	1.00	0.67	0.15	100.01	1.00	0.23	0.47	15,243.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	15,243	0	0.00	0.00	NaN	0.15	0.00	2.00	0.12	0.17	9,848.98	0.65	0.65	0.65
$seg_{\Delta t}$	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.11	0.15	10,747.99	0.71	0.71	0.71
$seg_{\Delta t-1}$	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.08	0.12	11,646.99	0.76	0.76	0.76
seg_{cent}	7,744	0	0.00	0.00	NaN	0.15	0.01	2.00	0.43	0.48	206.46	0.01	0.01	0.01
seg_{double}	15,243	15,243	0.50	1.00	0.67	0.15	100.00	1.00	0.02	0.45	14,343.99	0.47	0.94	0.63
seg_{shift}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	14,344.00	0.94	0.94	0.94
seg_{ten}	15,068	9,900	0.65	0.65	0.65	0.70	0.00	0.70	0.15	0.18	11,783.28	0.77	0.77	0.77
$\alpha = 0.5$														
seg_{const}	5,989	1,890	0.25	0.12	0.17	0.36	-50.00	1.25	0.48	0.49	3,205.91	0.42	0.21	0.28
seg_{rand}	4,049	913	0.12	0.06	0.08	0.31	-50.00	1.38	0.46	0.47	2,146.78	0.28	0.14	0.19
$\alpha = 1$														
seg_{const}	10,215	3,781	0.25	0.25	0.25	0.36	0.00	1.50	0.50	0.52	5,790.32	0.38	0.38	0.38
seg_{rand}	7,118	1,782	0.12	0.12	0.12	0.25	0.00	1.77	0.48	0.50	3,889.70	0.26	0.26	0.26
$\alpha = 2$														
seg_{const}	13,442	7,597	0.25	0.50	0.33	-0.09	100.00	2.00	0.55	0.60	9,097.36	0.30	0.60	0.40
seg_{rand}	10,780	3,593	0.12	0.24	0.16	-0.25	100.00	2.53	0.51	0.54	6,575.91	0.22	0.43	0.29

Table 4.5.: Results for dataset “MOCHA:fsew”. $\Delta t = 10\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	15,243.00	1.00	1.00	1.00
seg_{all}	15,243	15,243	0.02	1.00	0.03	-48.87	5,842.62	58.43	0.56	1.00	15,243.00	0.02	1.00	0.03
seg_{half}	7,622	7,622	1.00	0.50	0.67	0.65	-50.00	0.50	0.19	0.25	7,622.00	1.00	0.50	0.67
seg_{allC}	15,243	15,243	0.50	1.00	0.67	0.15	100.01	1.00	0.19	0.47	15,243.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	15,243	0	0.00	0.00	NaN	0.15	0.00	2.00	0.11	0.17	9,300.63	0.61	0.61	0.61
$seg_{\Delta t}$	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.10	0.15	10,291.03	0.68	0.68	0.68
$seg_{\Delta t-1}$	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.08	0.12	11,281.42	0.74	0.74	0.74
seg_{cent}	7,838	0	0.00	0.00	NaN	0.15	0.01	2.00	0.35	0.42	239.05	0.02	0.02	0.02
seg_{double}	15,243	15,243	0.50	1.00	0.67	0.15	100.00	1.00	0.02	0.44	14,252.60	0.47	0.94	0.62
seg_{shift}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	14,252.61	0.94	0.94	0.94
seg_{ten}	14,952	11,266	0.74	0.74	0.74	0.78	0.00	0.52	0.13	0.17	11,652.78	0.76	0.76	0.76
$\alpha = 0.5$														
seg_{const}	5,523	1,426	0.19	0.09	0.12	0.34	-50.00	1.31	0.47	0.49	3,014.59	0.40	0.20	0.26
seg_{rand}	3,545	853	0.11	0.06	0.07	0.31	-50.00	1.39	0.46	0.47	1,940.44	0.25	0.13	0.17
$\alpha = 1$														
seg_{const}	9,549	2,848	0.19	0.19	0.19	0.31	0.00	1.63	0.50	0.53	5,335.62	0.35	0.35	0.35
seg_{rand}	6,269	1,691	0.11	0.11	0.11	0.24	0.00	1.78	0.48	0.49	3,436.72	0.23	0.23	0.23
$\alpha = 2$														
seg_{const}	13,104	7,564	0.25	0.50	0.33	-0.09	100.00	2.01	0.56	0.62	8,609.47	0.28	0.56	0.38
seg_{rand}	9,636	3,341	0.11	0.22	0.15	-0.26	100.00	2.56	0.51	0.55	5,802.12	0.19	0.38	0.25

Table 4.6.: Results for dataset “MOCHA:msak”. $\Delta t = 10\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	11,639	11,639	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	11,639.00	1.00	1.00	1.00
seg_{all}	11,639	11,639	0.03	1.00	0.05	-28.49	3,454.41	34.54	0.50	1.00	11,639.00	0.03	1.00	0.05
seg_{half}	5,820	5,820	1.00	0.50	0.67	0.65	-50.00	0.50	0.25	0.25	5,820.00	1.00	0.50	0.67
seg_{allC}	11,639	11,639	0.50	1.00	0.67	0.15	100.01	1.00	0.35	0.51	11,639.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	11,562	41	0.00	0.00	0.00	0.15	0.00	1.99	0.30	0.30	7,030.64	0.60	0.60	0.60
$seg_{\Delta t}$	11,613	11,613	1.00	1.00	1.00	1.00	0.00	0.00	0.25	0.26	7,803.32	0.67	0.67	0.67
$seg_{\Delta t-1}$	11,627	11,627	1.00	1.00	1.00	1.00	0.00	0.00	0.20	0.21	8,569.61	0.74	0.74	0.74
seg_{cent}	5,973	41	0.00	0.00	0.00	0.15	0.01	1.99	0.69	0.70	128.45	0.01	0.01	0.01
seg_{double}	11,637	11,637	0.50	1.00	0.67	0.15	99.98	1.00	0.05	0.50	10,870.84	0.47	0.93	0.62
seg_{shift}	11,637	11,637	1.00	1.00	1.00	1.00	0.00	0.00	0.05	0.06	10,869.02	0.93	0.93	0.93
seg_{ten}	11,634	10,587	0.91	0.91	0.91	0.92	0.00	0.18	0.19	0.20	9,252.96	0.79	0.79	0.79
$\alpha = 0.5$														
seg_{const}	5,710	1,783	0.31	0.15	0.20	0.39	-50.00	1.19	0.50	0.50	2,878.99	0.49	0.25	0.33
seg_{rand}	4,796	1,493	0.26	0.13	0.17	0.37	-50.00	1.24	0.50	0.50	2,429.02	0.42	0.21	0.28
$\alpha = 1$														
seg_{const}	9,731	3,483	0.30	0.30	0.30	0.40	0.00	1.40	0.51	0.51	5,306.12	0.46	0.46	0.46
seg_{rand}	8,598	2,968	0.26	0.26	0.26	0.36	0.00	1.49	0.50	0.50	4,657.40	0.40	0.40	0.40
$\alpha = 2$														
seg_{const}	11,528	7,124	0.31	0.61	0.41	-0.03	100.00	1.78	0.50	0.50	8,226.59	0.35	0.71	0.47
seg_{rand}	11,158	6,010	0.26	0.52	0.34	-0.08	100.00	1.97	0.50	0.50	7,455.48	0.32	0.64	0.43

Table 4.7.: Results for dataset “BOSS:A”. $\Delta t = 10\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	2,292	2,292	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	2,292.00	1.00	1.00	1.00
seg_{all}	2,292	2,292	0.02	1.00	0.05	-33.70	4,065.10	40.65	0.50	1.00	2,292.00	0.02	1.00	0.05
seg_{half}	1,146	1,146	1.00	0.50	0.67	0.65	-50.00	0.50	0.25	0.25	1,146.00	1.00	0.50	0.67
seg_{allC}	2,292	2,292	0.50	1.00	0.67	0.15	100.04	1.00	0.36	0.50	2,292.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	2,282	9	0.00	0.00	0.00	0.15	0.00	1.99	0.26	0.27	1,535.61	0.67	0.67	0.67
$seg_{\Delta t}$	2,283	2,283	1.00	1.00	1.00	1.00	0.00	0.01	0.22	0.23	1,658.86	0.72	0.72	0.72
$seg_{\Delta t-1}$	2,285	2,285	1.00	1.00	1.00	1.00	0.00	0.01	0.18	0.18	1,783.68	0.78	0.78	0.78
seg_{cent}	1,172	9	0.00	0.00	0.00	0.15	0.04	1.99	0.72	0.72	20.15	0.01	0.01	0.01
seg_{double}	2,292	2,292	0.50	1.00	0.67	0.15	100.00	1.00	0.05	0.50	2,164.27	0.47	0.94	0.63
seg_{shift}	2,292	2,292	1.00	1.00	1.00	1.00	0.00	0.00	0.05	0.05	2,164.33	0.94	0.94	0.94
seg_{ten}	2,292	1,905	0.83	0.83	0.83	0.86	0.00	0.34	0.20	0.20	1,826.26	0.80	0.80	0.80
$\alpha = 0.5$														
seg_{const}	1,137	311	0.27	0.14	0.18	0.37	-50.00	1.23	0.50	0.50	589.26	0.51	0.26	0.34
seg_{rand}	997	238	0.21	0.10	0.14	0.35	-50.00	1.29	0.51	0.51	493.23	0.43	0.22	0.29
$\alpha = 1$														
seg_{const}	1,946	592	0.26	0.26	0.26	0.37	0.00	1.48	0.50	0.50	1,073.53	0.47	0.47	0.47
seg_{rand}	1,756	511	0.22	0.22	0.22	0.34	0.00	1.55	0.51	0.51	934.75	0.41	0.41	0.41
$\alpha = 2$														
seg_{const}	2,281	1,192	0.26	0.52	0.35	-0.08	100.00	1.96	0.50	0.50	1,641.77	0.36	0.72	0.48
seg_{rand}	2,232	1,051	0.23	0.46	0.31	-0.11	100.00	2.08	0.50	0.50	1,514.32	0.33	0.66	0.44

Table 4.8.: Results for dataset “BOSS:B”. $\Delta t = 10\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	110,443	110,443	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	110,443.00	1.00	1.00	1.00
seg_{all}	110,443	110,443	0.01	1.00	0.02	-77.47	9,193.71	91.94	0.64	1.00	110,443.00	0.01	1.00	0.02
seg_{half}	55,222	55,222	1.00	0.50	0.67	0.65	-50.00	0.50	0.12	0.24	55,222.00	1.00	0.50	0.67
seg_{allC}	110,443	110,443	0.50	1.00	0.67	0.15	100.00	1.00	0.05	0.38	110,443.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	95,515	17,361	0.16	0.16	0.16	0.28	0.00	1.69	0.05	0.17	38,326.81	0.35	0.35	0.35
$seg_{\Delta t}$	98,528	98,528	0.89	0.89	0.89	0.91	0.00	0.22	0.04	0.16	42,456.58	0.38	0.38	0.38
$seg_{\Delta t-1}$	102,150	102,150	0.92	0.92	0.92	0.94	0.00	0.15	0.04	0.15	47,785.43	0.43	0.43	0.43
seg_{cent}	56,452	15,330	0.14	0.14	0.14	0.26	0.00	1.72	0.09	0.23	1,256.50	0.01	0.01	0.01
seg_{double}	110,443	110,443	0.50	1.00	0.67	0.15	100.00	1.00	0.01	0.36	103,309.14	0.47	0.94	0.62
seg_{shift}	110,442	110,442	1.00	1.00	1.00	1.00	0.00	0.00	0.01	0.02	103,308.23	0.94	0.94	0.94
seg_{ten}	107,657	106,643	0.97	0.97	0.97	0.97	0.00	0.07	0.05	0.10	85,656.25	0.78	0.78	0.78
$\alpha = 0.5$														
seg_{const}	24,489	12,103	0.22	0.11	0.15	0.35	-50.00	1.28	0.43	0.46	13,552.96	0.25	0.12	0.16
seg_{rand}	11,160	4,654	0.08	0.04	0.06	0.30	-50.00	1.42	0.39	0.40	6,408.68	0.12	0.06	0.08
$\alpha = 1$														
seg_{const}	45,738	24,302	0.22	0.22	0.22	0.33	0.00	1.56	0.50	0.56	24,446.85	0.22	0.22	0.22
seg_{rand}	19,605	9,467	0.09	0.09	0.09	0.22	0.00	1.83	0.41	0.44	10,980.52	0.10	0.10	0.10
$\alpha = 2$														
seg_{const}	79,945	48,634	0.22	0.44	0.29	-0.12	100.00	2.12	0.63	0.76	44,275.25	0.20	0.40	0.27
seg_{rand}	34,468	19,043	0.09	0.17	0.11	-0.30	100.00	2.66	0.47	0.52	19,183.27	0.09	0.17	0.12

Table 4.9.: Results for dataset “IMS:ms”. $\Delta t = 20\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	90,936	90,936	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	90,936.00	1.00	1.00	1.00
seg_{all}	90,936	90,936	0.02	1.00	0.03	-48.36	5,782.72	57.83	0.54	1.00	90,936.00	0.02	1.00	0.03
seg_{half}	45,468	45,468	1.00	0.50	0.67	0.65	-50.00	0.50	0.21	0.25	45,468.00	1.00	0.50	0.67
seg_{allC}	90,936	90,936	0.50	1.00	0.67	0.15	100.00	1.00	0.20	0.48	90,936.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	80,871	10,710	0.12	0.12	0.12	0.25	0.00	1.76	0.22	0.27	35,312.17	0.39	0.39	0.39
$seg_{\Delta t}$	83,198	83,198	0.91	0.91	0.91	0.93	0.00	0.17	0.20	0.25	39,075.70	0.43	0.43	0.43
$seg_{\Delta t-1}$	85,264	85,264	0.94	0.94	0.94	0.95	0.00	0.12	0.18	0.24	43,338.44	0.48	0.48	0.48
seg_{cent}	46,363	9,757	0.11	0.11	0.11	0.24	0.00	1.79	0.38	0.43	928.33	0.01	0.01	0.01
seg_{double}	90,934	90,934	0.50	1.00	0.67	0.15	99.99	1.00	0.02	0.46	85,550.00	0.47	0.94	0.63
seg_{shift}	90,930	90,930	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	85,544.43	0.94	0.94	0.94
seg_{ten}	89,891	87,282	0.96	0.96	0.96	0.97	0.00	0.08	0.13	0.15	71,624.04	0.79	0.79	0.79
$\alpha = 0.5$														
seg_{const}	33,390	15,987	0.35	0.18	0.23	0.40	-50.00	1.15	0.48	0.49	17,588.72	0.39	0.19	0.26
seg_{rand}	18,200	8,306	0.18	0.09	0.12	0.34	-50.00	1.32	0.47	0.47	9,656.71	0.21	0.11	0.14
$\alpha = 1$														
seg_{const}	60,015	31,786	0.35	0.35	0.35	0.44	0.00	1.30	0.50	0.52	32,611.90	0.36	0.36	0.36
seg_{rand}	33,063	16,205	0.18	0.18	0.18	0.30	0.00	1.64	0.48	0.49	17,591.17	0.19	0.19	0.19
$\alpha = 2$														
seg_{const}	83,585	63,390	0.35	0.70	0.46	0.02	100.00	1.61	0.54	0.57	53,463.85	0.29	0.59	0.39
seg_{rand}	53,557	32,988	0.18	0.36	0.24	-0.17	100.00	2.27	0.50	0.52	31,212.55	0.17	0.34	0.23

Table 4.10.: Results for dataset “IMS:rk”. $\Delta t = 20$ ms

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	15,243.00	1.00	1.00	1.00
seg_{all}	15,243	15,243	0.02	1.00	0.03	-49.55	5,922.50	59.22	0.55	1.00	15,243.00	0.02	1.00	0.03
seg_{half}	7,622	7,622	1.00	0.50	0.67	0.65	-50.00	0.50	0.20	0.25	7,622.00	1.00	0.50	0.67
seg_{allC}	15,243	15,243	0.50	1.00	0.67	0.15	100.01	1.00	0.23	0.47	15,243.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	12,980	3,032	0.20	0.20	0.20	0.32	0.00	1.60	0.21	0.29	6,182.79	0.41	0.41	0.41
$seg_{\Delta t}$	12,705	12,705	0.83	0.83	0.83	0.86	0.00	0.33	0.20	0.27	6,249.45	0.41	0.41	0.41
$seg_{\Delta t-1}$	13,778	13,778	0.90	0.90	0.90	0.92	0.00	0.19	0.18	0.24	7,110.14	0.47	0.47	0.47
seg_{cent}	7,744	2,757	0.18	0.18	0.18	0.30	0.01	1.64	0.43	0.48	206.46	0.01	0.01	0.01
seg_{double}	15,243	15,243	0.50	1.00	0.67	0.15	100.00	1.00	0.02	0.45	14,343.99	0.47	0.94	0.63
seg_{shift}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	14,344.00	0.94	0.94	0.94
seg_{ten}	15,068	14,237	0.93	0.93	0.93	0.94	0.00	0.13	0.15	0.18	11,783.28	0.77	0.77	0.77
$\alpha = 0.5$														
seg_{const}	5,989	2,938	0.39	0.19	0.26	0.42	-50.00	1.11	0.48	0.49	3,205.91	0.42	0.21	0.28
seg_{rand}	3,968	1,622	0.21	0.11	0.14	0.35	-50.00	1.29	0.46	0.47	2,125.29	0.28	0.14	0.19
$\alpha = 1$														
seg_{const}	10,215	5,924	0.39	0.39	0.39	0.48	0.00	1.22	0.50	0.52	5,790.32	0.38	0.38	0.38
seg_{rand}	7,150	3,335	0.22	0.22	0.22	0.33	0.00	1.56	0.48	0.49	3,937.02	0.26	0.26	0.26
$\alpha = 2$														
seg_{const}	13,442	11,829	0.39	0.78	0.52	0.05	100.00	1.45	0.55	0.60	9,097.36	0.30	0.60	0.40
seg_{rand}	10,634	6,481	0.21	0.43	0.28	-0.13	100.00	2.15	0.52	0.55	6,469.54	0.21	0.42	0.28

Table 4.11.: Results for dataset “MOCHA:fsew”. $\Delta t = 20$ ms

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	15,243.00	1.00	1.00	1.00
seg_{all}	15,243	15,243	0.02	1.00	0.03	-48.87	5,842.62	58.43	0.56	1.00	15,243.00	0.02	1.00	0.03
seg_{half}	7,622	7,622	1.00	0.50	0.67	0.65	-50.00	0.50	0.19	0.25	7,622.00	1.00	0.50	0.67
seg_{allC}	15,243	15,243	0.50	1.00	0.67	0.15	100.01	1.00	0.19	0.47	15,243.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	12,555	3,784	0.25	0.25	0.25	0.36	0.00	1.50	0.19	0.27	5,699.18	0.37	0.37	0.37
$seg_{\Delta t}$	12,197	12,197	0.80	0.80	0.80	0.83	0.00	0.40	0.18	0.26	5,634.44	0.37	0.37	0.37
$seg_{\Delta t-1}$	13,375	13,375	0.88	0.88	0.88	0.90	0.00	0.25	0.16	0.24	6,392.79	0.42	0.42	0.42
seg_{cent}	7,838	3,426	0.22	0.22	0.22	0.34	0.01	1.55	0.35	0.42	239.05	0.02	0.02	0.02
seg_{double}	15,243	15,243	0.50	1.00	0.67	0.15	100.00	1.00	0.02	0.44	14,252.60	0.47	0.94	0.62
seg_{shift}	15,243	15,243	1.00	1.00	1.00	1.00	0.00	0.00	0.02	0.03	14,252.61	0.94	0.94	0.94
seg_{ten}	14,952	14,194	0.93	0.93	0.93	0.94	0.00	0.14	0.13	0.17	11,652.78	0.76	0.76	0.76
$\alpha = 0.5$														
seg_{const}	5,523	2,548	0.33	0.17	0.22	0.40	-50.00	1.17	0.47	0.49	3,014.59	0.40	0.20	0.26
seg_{rand}	3,561	1,594	0.21	0.10	0.14	0.35	-50.00	1.29	0.46	0.47	1,970.25	0.26	0.13	0.17
$\alpha = 1$														
seg_{const}	9,549	5,164	0.34	0.34	0.34	0.44	0.00	1.32	0.50	0.53	5,335.62	0.35	0.35	0.35
seg_{rand}	6,235	3,107	0.20	0.20	0.20	0.32	0.00	1.59	0.48	0.50	3,445.88	0.23	0.23	0.23
$\alpha = 2$														
seg_{const}	13,104	11,659	0.38	0.76	0.51	0.05	100.00	1.47	0.56	0.62	8,609.47	0.28	0.56	0.38
seg_{rand}	9,605	6,007	0.20	0.39	0.26	-0.15	100.00	2.21	0.51	0.55	5,717.26	0.19	0.38	0.25

Table 4.12.: Results for dataset “MOCHA:msak”. $\Delta t = 20\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	11,639	11,639	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	11,639.00	1.00	1.00	1.00
seg_{all}	11,639	11,639	0.03	1.00	0.05	-28.49	3,454.41	34.54	0.50	1.00	11,639.00	0.03	1.00	0.05
seg_{half}	5,820	5,820	1.00	0.50	0.67	0.65	-50.00	0.50	0.25	0.25	5,820.00	1.00	0.50	0.67
seg_{allC}	11,639	11,639	0.50	1.00	0.67	0.15	100.01	1.00	0.35	0.51	11,639.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	10,139	1,575	0.14	0.14	0.14	0.26	0.00	1.73	0.50	0.51	3,526.78	0.30	0.30	0.30
$seg_{\Delta t}$	10,484	10,484	0.90	0.90	0.90	0.92	0.00	0.20	0.46	0.47	4,043.68	0.35	0.35	0.35
$seg_{\Delta t-1}$	10,933	10,933	0.94	0.94	0.94	0.95	0.00	0.12	0.42	0.43	4,741.46	0.41	0.41	0.41
seg_{cent}	5,973	1,424	0.12	0.12	0.12	0.25	0.01	1.76	0.69	0.70	128.45	0.01	0.01	0.01
seg_{double}	11,637	11,637	0.50	1.00	0.67	0.15	99.98	1.00	0.05	0.50	10,870.84	0.47	0.93	0.62
seg_{shift}	11,637	11,637	1.00	1.00	1.00	1.00	0.00	0.00	0.05	0.06	10,869.02	0.93	0.93	0.93
seg_{ten}	11,634	11,579	0.99	0.99	0.99	1.00	0.00	0.01	0.19	0.20	9,252.96	0.79	0.79	0.79
$\alpha = 0.5$														
seg_{const}	5,710	3,372	0.58	0.29	0.39	0.49	-50.00	0.92	0.50	0.50	2,878.99	0.49	0.25	0.33
seg_{rand}	4,762	2,796	0.48	0.24	0.32	0.45	-50.00	1.02	0.50	0.50	2,394.48	0.41	0.21	0.27
$\alpha = 1$														
seg_{const}	9,731	6,522	0.56	0.56	0.56	0.62	0.00	0.88	0.51	0.51	5,306.12	0.46	0.46	0.46
seg_{rand}	8,578	5,606	0.48	0.48	0.48	0.56	0.00	1.04	0.50	0.50	4,618.68	0.40	0.40	0.40
$\alpha = 2$														
seg_{const}	11,528	11,528	0.50	0.99	0.66	0.14	100.00	1.02	0.50	0.50	8,226.59	0.35	0.71	0.47
seg_{rand}	11,202	10,325	0.44	0.89	0.59	0.10	100.00	1.23	0.50	0.50	7,456.54	0.32	0.64	0.43

Table 4.13.: Results for dataset “BOSS:A”. $\Delta t = 20\,\text{ms}$

	aligned	TP	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{TP}$	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
seg_{ref}	2,292	2,292	1.00	1.00	1.00	1.00	0.00	0.00	0.00	0.00	2,292.00	1.00	1.00	1.00
seg_{all}	2,292	2,292	0.02	1.00	0.05	-33.70	4,065.10	40.65	0.50	1.00	2,292.00	0.02	1.00	0.05
seg_{half}	1,146	1,146	1.00	0.50	0.67	0.65	-50.00	0.50	0.25	0.25	1,146.00	1.00	0.50	0.67
seg_{allC}	2,292	2,292	0.50	1.00	0.67	0.15	100.04	1.00	0.36	0.50	2,292.00	0.50	1.00	0.67
$seg_{\Delta t+1}$	2,147	126	0.05	0.05	0.05	0.19	0.00	1.89	0.45	0.46	905.25	0.39	0.39	0.39
$seg_{\Delta t}$	2,193	2,193	0.96	0.96	0.96	0.96	0.00	0.09	0.42	0.42	1,022.03	0.45	0.45	0.45
$seg_{\Delta t-1}$	2,232	2,232	0.97	0.97	0.97	0.98	0.00	0.05	0.38	0.39	1,149.02	0.50	0.50	0.50
seg_{cent}	1,172	122	0.05	0.05	0.05	0.19	0.04	1.89	0.72	0.72	20.15	0.01	0.01	0.01
seg_{double}	2,292	2,292	0.50	1.00	0.67	0.15	100.00	1.00	0.05	0.50	2,164.27	0.47	0.94	0.63
seg_{shift}	2,292	2,292	1.00	1.00	1.00	1.00	0.00	0.00	0.05	0.05	2,164.33	0.94	0.94	0.94
seg_{ten}	2,292	2,275	0.99	0.99	0.99	0.99	0.00	0.01	0.20	0.20	1,826.26	0.80	0.80	0.80
$\alpha = 0.5$														
seg_{const}	1,137	580	0.51	0.25	0.34	0.46	-50.00	0.99	0.50	0.50	589.26	0.51	0.26	0.34
seg_{rand}	994	509	0.44	0.22	0.30	0.44	-50.00	1.06	0.50	0.50	494.46	0.43	0.22	0.29
$\alpha = 1$														
seg_{const}	1,946	1,132	0.49	0.49	0.49	0.57	0.00	1.01	0.50	0.50	1,073.53	0.47	0.47	0.47
seg_{rand}	1,777	978	0.43	0.43	0.43	0.51	0.00	1.15	0.50	0.50	953.22	0.42	0.42	0.42
$\alpha = 2$														
seg_{const}	2,281	2,281	0.50	1.00	0.66	0.14	100.00	1.01	0.50	0.50	1,641.77	0.36	0.72	0.48
seg_{rand}	2,232	1,985	0.43	0.87	0.58	0.09	100.00	1.27	0.50	0.50	1,518.27	0.33	0.66	0.44

Table 4.14.: Results for dataset “BOSS:B”. $\Delta t = 20$ ms

evaluation measure has the property that the perfect segmentation, which is defined as $hyp(seg_{\text{ref}}) = ref$, gets the highest score in comparison to any other possible segmentation.

(B) Test whether $s(hyp(seg_{\text{shift}}), ref) < s(hyp(seg_{\text{ref}}), ref)$. This holds if an evaluation measure is sensitive to near misses: A minimum deviance of a hypothesised boundary from the expected reference location (as generated by seg_{shift}) should be reflected by a lower score.

(C) Test whether $s(hyp(seg_{\text{ref}}), ref)$ gets a different score than $s(hyp(seg_{\Delta t-1}, ref)$. This holds only for those measures which do not employ a tolerance window around true segment locations. If a hypothesised boundary is just within such a fixed distance from the reference location, the score should still be lower than that of a perfect segmentation.

(D) Test whether $s(hyp(seg_{\text{ref}}), ref)$ gets a different score from $s(hyp(seg_{\text{cent}}, ref)$. This is true for any evaluation measure that considers not only the total number of boundaries but also their locations.

(E) Test whether $s(hyp(seg_{\text{shift}}), ref) > s(hyp(seg_{\text{double}}), ref)$, i.e., whether the segmentation generated by seg_{shift} gets a higher score than that generated by seg_{double}. Half of the segment boundaries produced by seg_{double} are false alarms, and this, in general, should be reflected by a lower score in comparison to the segmentation produced by seg_{shift}. The biased segmenter seg_{shift} that just shifts all boundaries by one frame does not introduce false boundaries, while seg_{double} produces a massively over-segmented result.

In practice, however, a segmentation method could incorporate an additional processing step which merges boundaries that are very close to each other. Equivalently, a minimum required distance might be introduced in β which maps the continuous boundariness values b to hard segment boundaries[20]. A segmenter's constant bias in boundary placement for different transition types, on the other hand, can also be compensated for (cf. Ljolje et al., 1997, p.309f).

The tests listed above are probably valid for most segmentation tasks. I add another set of tests which are debatable with respect to the actual requirements of a segmentation. These tests are specified as follows:

[20] In Scharenborg et al.'s (2010, p.1087) experiments, for example, the boundariness function b is the Euclidean distance between cluster means and the discretising segmentation function β is realised by a peak-picking algorithm. The latter is tuned by a parameter which implicitly controls the number of generated boundaries. Additionally, they apply a "smoothing" on the boundaries which are generated based on this cluster distance and the boundaries which are generated based on a separate pattern matching step: if the two methods generate two boundaries within a distance of 10 ms, the two boundaries are merged into one (the experiments of Scharenborg et al., 2010, are discussed in section 4.1.3).

(F) Test whether $s(hyp(seg_{\text{ref}}), ref)$ gets a score which is substantially higher than the scores for any other segmentation. This is an addition to test (A). Table 4.7, for example, shows an F_1 value of 1.00 for seg_{ref}, as well as for $seg_{\Delta t-1}$ and seg_{shift}. Test (A) is still satisfied by F_1, since the actual values for $seg_{\Delta t-1}$ and seg_{shift} are around 0.998262 and 0.999991, respectively. Due to rounding to two decimal digits this difference is not shown in the table. Test (F) therefore assesses whether the difference between the perfect score and the scores for the remaining segmentations is significant. Accordingly, F_1 in table 4.7 does not satisfy test criterion (F).

(G) Test whether $s(hyp(seg_{\text{all}}), ref)$ gets a score lower than any other segmentation. This assumes, that a maximally segmented sequence is the worst possible solution (among the diagnostic segmenters defined here).

An alternative definition of "the worst solution" would be an unsegmented sequence without any internal boundaries, but this was omitted from the current experiments.

(H) Test whether $s(hyp(seg_{\text{all}}), ref)$ gets a score which is substantially lower than the scores for any other segmentation.

(I) Test whether $s(hyp(seg_{\text{ref}}), ref) > s(hyp(seg_{\text{allC}}), ref)$. It can be observed that some evaluation measures (e.g. recall) assign the same score to seg_{allC} as they do for the perfect segmentation. This should not be the case.

(J) Test whether $s(hyp(seg_{\text{allC}}), ref)$ gets a higher score than any of the other segmenters which produce twice the true number of boundaries, i.e. seg_{double} and the baseline segmentations seg_{const} and seg_{rand} with $\alpha = 2$. This is motivated by the fact that half of the boundaries in seg_{allC} are exactly at the correct location, while this is not true for the other segmenters.

(K) Test whether $s(hyp(seg_{\text{ref}}), ref)$ gets a score which is substantially higher than the score for $seg_{\Delta t-1}$. This is an addition to (F) targeting specifically at evaluation measures which employ a fixed tolerance window.

The tests (A)–(K) are defined *within* a given result set. The two different settings for the tolerance window size parameter make it possible to compare the results *across* result sets. The following tests are applied to the evaluations of segmentations of the same data set with Δt set to 10 ms and 20 ms:

(L) Test whether $s(hyp(seg_{\text{cent}}), ref)$ is assigned the same score by an evaluation measure for both window sizes.

The segmentation generated by $s(hyp(seg_{\text{cent}}), ref)$ is the same for any given value of Δt. This test targets the independence of an evaluation measure from a fixed tolerance window.

(M) Test whether $s(hyp(seg_{\Delta t-1,\Delta t=10}, ref) > s(hyp(seg_{\Delta t-1,\Delta t=20}, ref)$.

Since the segmentation produced by $seg_{\Delta t-1}$ depends on the tolerance window size parameter, the assigned score should be higher for the smaller shift with $\Delta t = 10$ ms in comparison to the larger shift with $\Delta t = 20$ ms.

(N) Test whether the baseline segmentations on average get a score which is not significantly different for different tolerance window sizes.

It could also be tested whether the score for seg_{shift} is higher than the score for $seg_{\Delta t-1}$. Since both segmenters produce boundaries which are within the tolerance window of the reference, both should be, in general, correct. However, as a comparison of the matched boundaries shows for these two segmenters, the alignment is changed by the larger shift. This is due to the fact that some segments are shorter than the tolerance window. A tolerance window of $\Delta t = 10$ ms, for example, covers 10.76% of all frames in the IMS ms data set, and a tolerance window of $\Delta t = 20$ ms covers 21.12% of all frames.

Every test condition that evaluates to 'true' is assigned one scoring point. The individual scores are summed up to get a total score for each of the evaluation measures (see table 4.15 on page 136 for a summary of the results). The highest scoring measure can then be interpreted as the most appropriate evaluation measure for the task of speech segmentation according to the desired properties defined by the test conditions (A)–(K).

Given the same segmentations, the measures precision, recall, F_1, R, and S' all improve with increasing tolerance, as can be seen by comparing tables 4.3–4.8 for $\Delta t = 10$ ms with the corresponding tables 4.9–4.14 for $\Delta t = 20$ ms. A perfect score for $seg_{\Delta t-1}$ could be expected for the tolerance-window-based measures: Although all boundaries are displaced, they are still within tolerance. F_1-measure in tables 4.3–4.8, for example, gives identical results for $seg_{\Delta t-1}$ and seg_{shift} with $\Delta t = 10$ ms (all boundaries are within tolerance in both segmentations). Increasing the tolerance from ± 10 ms to ± 20 ms, however, causes the score of $seg_{\Delta t-1}$ to decrease, as shown in tables 4.9–4.14. This can be explained by the fact that a tolerance of ± 20 ms corresponds to half the mean segment length of the reference data. Thus, the alignment is affected significantly, as, after shifting all boundaries by 18 ms (= 20 ms − 1 frame), some hypothesised boundaries happen to be closer to the next reference than to their original counterpart. However, $seg_{\Delta t-1}$ is in any case worse than seg_{shift}, since the distance to the reference is much larger for every hypothesised boundary. With $\Delta t = 10$ ms, this difference is reflected only by P_k, *WindowDiff* and the measures based on $\widetilde{TP}$, and in one case also by S' in table 4.8. Compare also OS and S', which generally fail to recognise the errors of $seg_{\Delta t-1}$

and seg_{shift} for $\Delta t = 10$ due to the correct total number of boundaries. With $\Delta t = 20$, however, S' successfully detects that there is something wrong with the segmentations generated by $seg_{\Delta t-1}$.

Placing hypothesised boundaries consistently at the centre of each true segment, as seg_{cent} does, makes each hypothesised boundary equally bad. Yet, the evaluation measures which depend on a tolerance window give different results for $\Delta t = 10$ ms and $\Delta t = 20$ ms. I consider this dependency on a fixed Δt a major problem for these evaluation measures. They treat all discrepancies between hypothesis and reference alike if they are within the defined tolerance window, even though larger deviations from the gold standard should always be penalised more than minor deviations. The degree of error is, e.g., comparable for deviations of 21 ms and 19 ms, and both are much worse than a deviation of 1 ms from a reference boundary. With a tolerance of ±20 ms, however, the boundary with a 21 ms deviation would be counted as wrong while the latter two are counted equally correct. However, the actual size of the tolerance window in comparison to the sizes of the segments disturbs the theoretically expected results. The alignment decreases *TP* of $seg_{\Delta t-1}$ as compared to seg_{shift} since some of the shifted boundaries are closer to the following reference.

P_k and *WindowDiff* are, in this respect, better evaluation measures as they don't use a fixed tolerance window around each reference boundary to divide the hypothesised boundaries into hits and non-hits. Still, P_k shows no difference between seg_{double} and seg_{shift}, while *WindowDiff* reflects the fact that seg_{shift} is intuitively much better than seg_{double}, with respect to over-segmentation. P_k has the same values for seg_{const}, seg_{rand}, seg_{all} and seg_0 — even though seg_{all} is clearly worse than any other segmentation as far as over-segmentation is concerned. *WindowDiff* appropriately gives a much lower score for seg_{all}, highlighting the benefit of using a measure which is independent of the tolerance window and which considers the number of boundaries inside the evaluation window.

Conclusions on evaluation measures

There are different types of errors. An evaluation measure should reflect all relevant error types, and it should be clear which errors are considered *relevant* by a given measure and how they are weighted. An evaluation measure should be independent of specific properties of the speech material. It should penalise larger deviations more than smaller ones. Exact hits are very unlikely in speech segmentation and deviations from a reference cannot be simply regarded as errors due to the inherent acoustic variability of speech and the common disagreement between humans about the exact locations of segment boundaries (cf. Wesenick and Kipp, 1996, p.129ff). Using a manually created reference with a tolerance window approach is commonly the preferred way to tackle this evaluation problem. However, this does not consider various degrees of deviation from the reference appropriately. Treatment of special cases (like overlapping tolerance windows, hypothesised boundaries within stretches of silence, the existence of two

nearest references etc.) should be consistent and should be reported to make the evaluation results comparable and reproducible. If precision, recall and F-measure are needed in order to compare results with other studies, TP should be determined according to the proposed alignment criteria in definition 4.1. This treats the alignment problems associated with a tolerance window approach in a consistent way, it is easy to apply and to reproduce, and it makes results comparable.

A range of diagnostic segmentation methods was implemented to help understand certain properties of the various evaluation measures and examine their systematic strengths and weaknesses. The performance of these measures was compared under a variety of different conditions. Two new performance measures for speech segmentation were proposed in section 4.2: the modified slot error rate S' and the count of relative or fuzzy true positives $\widetilde{TP}$ as a basis for a locally-sensitive F_1-measure $\widetilde{F}_1$. Both measures outperform the classic F-measure as shown by the scores on the tests summarised in table 4.15 (on page 136).

The evaluation measures R and S' show a sensitivity to the size of the tolerance window in the experiments reported in this section. However, they could also be redefined based on a different definition of true positives, similarly to $\widetilde{F}_1$. How such redefined measures would perform in comparison to the other measures which are discussed here remains to be investigated. Comparing the overall scores of F_1 and $\widetilde{F}_1$ in table 4.15 indicates that there might be quite a significant improvement to be gained.

One conclusion which can be drawn from the experiments presented in this section is, that *WindowDiff* and $\widetilde{F}_1$ are the best measures for the evaluation of speech segmentation solutions. Given their superior ability compared to P_k and the F-measure as well as the S' measure in handling near misses, both *WindowDiff* and $\widetilde{F}_1$ can be considered better direct evaluation measures for automatic speech segmentation. No attempt was made at formally analysing the questions which are addressed in this section with mathematical rigour. On the one hand, this would be beyond the scope of this thesis. On the other hand, the approach taken here is consistent with the spirit of this thesis in using simulation experiments. This section also demonstrated how an existing measure can be adjusted to improve its performance with respect to some intuitively appropriate, or desired properties expected from an evaluation measure.

Test data	prc	rec	F_1	R	OS	S'	P_k	WD	$\widetilde{\text{prc}}$	$\widetilde{\text{rec}}$	$\widetilde{F}_1$
IMS:ms ($\Delta t = 10\,\text{ms}$)	6	4	7	8	5	8	7	10	8	6	10
IMS:ms ($\Delta t = 20\,\text{ms}$)	7	5	8	9	5	9	7	10	8	6	10
IMS:ms	1	1	1	1	2	1	3	3	3	3	3
IMS:rk ($\Delta t = 10\,\text{ms}$)	7	5	8	8	5	9	8	10	8	6	10
IMS:rk ($\Delta t = 20\,\text{ms}$)	8	6	9	9	5	10	8	10	8	6	10
IMS:rk	1	1	1	1	2	1	3	3	3	3	3
MOCHA:fsew ($\Delta t = 10\,\text{ms}$)	4	2	4	5	5	5	9	10	8	7	10
MOCHA:fsew ($\Delta t = 20\,\text{ms}$)	6	4	6	7	5	7	9	10	8	7	10
MOCHA:fsew	1	1	1	1	2	1	3	3	3	3	3
MOCHA:msak ($\Delta t = 10\,\text{ms}$)	4	2	4	5	5	5	9	10	8	6	10
MOCHA:msak ($\Delta t = 20\,\text{ms}$)	6	4	6	7	5	7	9	10	8	6	10
MOCHA:msak	1	1	1	1	2	1	3	3	3	3	3
BOSS:A ($\Delta t = 10\,\text{ms}$)	7	5	8	8	5	9	7	10	8	6	10
BOSS:A ($\Delta t = 20\,\text{ms}$)	8	6	9	9	5	10	7	10	8	6	10
BOSS:A	1	1	1	1	2	1	3	3	3	3	3
BOSS:B ($\Delta t = 10\,\text{ms}$)	5	3	5	6	5	6	7	10	8	7	10
BOSS:B ($\Delta t = 10\,\text{ms}$)	6	4	6	7	5	7	7	10	8	7	10
BOSS:B	1	1	1	1	2	1	3	3	3	3	3
Total	80	56	86	94	72	98	112	138	114	94	138

Table 4.15.: Summary of scores for evaluation measures. For details see page 118.

4.4. Experiment: Whole-sequence segmentation

This section presents a series of simulation experiments on unsupervised speech segmentation in the context of language acquisition. The segmentation experiments are directly motivated by observations made by Wade et al. (2010) which they made in relation to a simulation experiment on a memory-based speech production model — the Context Sequence Model (CSM)[21]. The model selects speech segments from a sequential memory by considering the context surrounding the respective candidate segments in memory. An experiment was carried out to determine the amount of context that provides useful information for the selection of segments from the memory. The length of the context was varied systematically from 0 to 1 s and a global match score was computed according to context size. The approach can be summarised as follows: A probe is constructed which consists of the selected segment plus preceding and following context. The probe is then compared with the entire memory sequence by computing the cross-correlation series between the probe signal and the memory. Wade et al. discuss their findings and speculate about a possible speech segmentation method based on their approach of direct comparisons of a probe with an entire sequential memory:

> "Some pilot experiments suggest that a useful segmentation approach may follow straightforwardly from the types of direct signal envelope comparisons described in Experiment 1. We have considered match sequences [...] where the total match of a probe sequence with an entire memory is measured as a function of the length of the sequence as more and more information is added to its right side. The derivative of such a function can be thought of as a measure of how likely or predictable each bit of new information is given the preceding portion of the probe sequence, and its inverse might be considered a measure of 'boundariness,' or how likely each point is to represent a juncture between functional units with separate category labels. [...] Further research will be required to determine whether this type of segmentation, along with a complementary categorization process, could result in a comprehensive Context Sequence-like account of phonological, and perhaps higher-level, processes." (Wade et al., 2010, p.237)

One particular problem of the original study by Wade et al. (2010) for a model of speech segmentation is their incorporation of segmental information. The experiments presented in this section therefore adopt the basic idea of a long, sequential memory of acoustic speech data but assume no given structural or segmental information. The approach of comparing short stretches of speech with an entire sequential memory is adopted as well. The hypothesis is that, as more context is added to a short stretch of speech, there will be increasingly fewer similar stretches of equal length in the memory. If the accumulated similarity is computed as a function of the probe length, some effects are expected to become visible as the probe length crosses a linguistic

[21] The CSM and the experiments of Wade et al. (2010) are discussed in detail in chapter 6, where I also present a series of experiments which extend the original work.

boundary. For example, if the probe is extended beyond the length of the corresponding phonetic segment, there will be only as many similar stretches in the memory as there are collocations of the same segment types. Below that threshold, a comparison might find all uni-gram segments of that type.

This reasoning is also supported by findings from a study by Johnson (1997b). In that study, speech is assumed to be represented by auditory exemplars which do not have an explicit internal structure (p.104). Recognition of a new word is based on activation of word models stored in memory. Differences in the activation patterns over time show emergent segmental structure within the whole-word exemplars (p.108)[22]. Based on the observations by Johnson (1997b) and Wade et al. (2010), characteristic patterns are expected to be found by a comparison of an item against an entire sequential memory in the experiments presented here.

The remainder of this section is organised as follows: First, two experiments on artificial data are presented in sections 4.4.1 and 4.4.2. Section 4.4.3 presents baseline results for experiments 1 and 2. The developed approach is then applied to real speech data in section 4.4.4. In each sub-section, the method and the details of the implementation are described and some representative results are shown. A discussion in 4.4.5 concludes this section.

4.4.1. Experiment 1: Constant-shift artificial data

The first experiment uses artificially constructed input data which are referred to as *constant-shift corpora*. They are constructed to make the segmentation task easy while still providing useful insight. The aim is to gain insight into the segmentation method under controlled data conditions. Although artificially constructed, the input sequences are referred to as "corpora" in order to emphasise the intended interpretation within the scope of the experiments presented here.

The following notation is used in this section for the main experimental parameters and the examined objects:

Notation	*Meaning*
$\mathcal{Y}$	the corpus, referred to as the memory sequence — a long sequential memory of speech items perceived in the past
$\mathcal{X}$	the probe — a short interval of speech that is to be segmented
N	the total length of the corpus
D	the number of dimensions of the corpus and the probes
T_p	the maximum length of a probe

[22]See section 4.1.4 on page 95 for more details.

...continued from previous page:

Notation	*Meaning*
l_t	the length of the target segment
l_p	the length of a probe (usually with $1 \leq l_p \leq T_p$)
l_Δ	the distance between two successive occurrences of the target segment in the corpus, referred to as the *shift*

Using this notation, the *speech segmentation problem* is formally defined as follows (cf. Duran et al., 2010b, section 3): A segmentation of a given probe $\mathcal{X}$ of length l_p is a sequence of boundaries $0 < t_1 < t_2 < \cdots < t_n < l_p$, with $n \geq 1$. The segmentation boundaries are determined based on a computational analysis of the distribution of the acoustic sounds of $\mathcal{X}$ in a large memory $\mathcal{Y}$ of experienced speech items. This definition of the problem contrasts with usual approaches to speech segmentation as discussed in section 4.1. Here, speech is not represented in a discrete alphabet. No a priori defined linguistic structure is assumed for the individual speech items (the probe $\mathcal{X}$) or the memory. The aim of this experiment and the following ones presented in this section is to investigate how segmental structure can be determined in an unsupervised fashion. This is a fundamental problem that needs to be solved in first language acquisition.

Method and implementation

The segmentation experiment is implemented and executed in MATLAB. The script `runExperiment1` shown in source code listing B.24 on page 300 defines the experimental parameters. It starts the main experiment function[23] `experiment1` shown in listing B.26 on page 303. The script file `bndSettings` shown in listing B.25 on page 302 defines some shared parameters (basically for the production of graphs), which are also used by the two experiments discussed later in this chapter.

The function `experiment1` first loads the shared parameters from `bndSettings`, and initialises the environment (the output directory, the logging mechanism etc.). The artificial test data is then created according to the given specifications by the function `createConstantShiftMS` shown in listing B.27 on page 306. The constructed corpus $\mathcal{Y}$ consists of a sequence of D-dimensional, length-normalised random vectors. A target segment of length l_t is created according to the parameter specification. It is repeated in the corpus $\mathcal{Y}$ with a constant distance l_Δ between the onset of two successive occurrences. These randomly generated corpora serve as the memory sequence for a particular run of

[23]The term "function" is used in this section to refer to MATLAB program functions. These are named software blocks that can take some input arguments, do some processing, and optionally return some output arguments. No explicit distinction is made in the remaining text between software functions and functions in a mathematical sense if it is clear from the context which term is meant.

the experiment. The full-length probe $\mathcal{X}$ for the experiment is generated based on the target segment, padding it with additional random vectors to a standard length of T_p frames. The function `createProbe` generates such a full-length probe (see listing B.28).

The expectation from such a set-up was that it should be easy to segment the probe $\mathcal{X}$ into the target segment with length l_t and its second, randomly generated part. Assuming that the random number generator is implemented reasonably[24], the only expected structure within the memory sequence is the repeated target segment at constant intervals separated by random noise. There are many identical instances of the target in the corpus and the possible continuations after their last frames are different for (probably) all instances. Moreover, there will be no exact match for the full length probe in the memory sequence if $l_t < T_p$.

A number of similarity measures for a given memory sequence and a probe is then computed by the function `getSimilarity`. This is implemented in C and incorporated as a binary `mex` file into the MATLAB code[25]. The C source code is shown in listing B.29 on page 307. As can be seen in the source code, `getSimilarity` is a collection of functions. The interface to MATLAB is the function `mexFunction`. This takes the input data from the running MATLAB scripts, starts the actual computations and sets the results. The main motivation for implementing the similarity function in C was to speed up the very expensive computations by using OPENMP[26] which allows parallel processing.

The similarity measures are based on the correlation score between a prefix of the probe and a stretch of the corpus of the same length. The raw correlation score r between a prefix of the probe $\mathcal{X}$ and a stretch of $\mathcal{Y}$ from index n to index $m = n + l_p - 1$ is computed according to equation 4.16. Additionally, the normalised correlation score ρ is computed according to equation 4.17.

$$r(\mathcal{X}, \mathcal{Y}, n, l_p) = \sum_{d=1}^{D} \sum_{i=1}^{l_p} x_{i,d} y_{i+n-1,d} \tag{4.16}$$

$$\rho(\mathcal{X}, \mathcal{Y}, n, l_p) = \sum_{d=1}^{D} \frac{\sum_{i=1}^{|\mathcal{X}|} x_{i,d} y_{i+n-1,d}}{\sqrt{\sum_{i=1}^{l_p} x_{i,d}^2} \sqrt{\sum_{i=1}^{l_p} y_{i+n-1,d}^2}} \tag{4.17}$$

where $x_{i,d}$ is the dth component (dimension) of the ith vector of the probe $\mathcal{X}$, and $y_{j,d}$ is the dth component of the jth vector of the memory sequence $\mathcal{Y}$. A global similarity

[24] Here, MATLAB's `randn` function is used which generates normally distributed pseudo-random numbers.

[25] MEX stands for "MATLAB executable". This provides an interface to subroutines implemented in C (or other programming languages). The C code is compiled into a binary file which is loaded and executed by MATLAB.

[26] `http://openmp.org` (accessed 2010-08)

score for a given probe $\mathcal{X}$ of length $1 < l_p < T_p$ and the entire memory sequence $\mathcal{Y}$ is computed by accumulating the correlation scores for $f = r$ or for $f = \rho$ as follows:

$$C(\mathcal{X}, \mathcal{Y}, l_p) = \sum_{i=1}^{N-l_p+1} f(\mathcal{X}, \mathcal{Y}, i, l_p) \tag{4.18}$$

The derivative of the global similarity score as defined in equation 4.18 indicates how much additional similarity can be found when the prefix of $\mathcal{X}$ is gradually extended:

$$C'(\mathcal{X}, \mathcal{Y}, l_p) = C(\mathcal{X}, \mathcal{Y}, l_p) - C(\mathcal{X}, \mathcal{Y}, l_p - 1) \qquad (\text{with} \quad 1 \leq l_p \leq T_p) \tag{4.19}$$

The original hypothesis was that minima in C' as defined in equation 4.19 would indicate locations of linguistically plausible segmentation boundaries (cf. Wade et al., 2010, p.237 and Duran et al., 2010b, p.149f). We found that this is not the case. Especially when real speech data is used, most of the sub-sequences in a corpus are not similar to a given probe and (intuitively) not relevant for its segmentation.

Additional similarity measures are therefore defined which consider only parts of the memory sequence with a minimum amount of similarity. A similarity threshold value θ is defined below which a sub-sequence of $\mathcal{Y}$ is not considered for the global similarity score. The threshold is defined for a correlation series based on its mean and the standard deviation. The mean μ and standard deviation σ for a set of values S are estimated as follows:

$$\mu(S) = \frac{\sum_{v \in S} v}{|S|} \tag{4.20}$$

$$\sigma(S) = \sqrt{\frac{\sum_{v \in S} (v - \mu(S))^2}{|S| - 1}} \tag{4.21}$$

Since high values of the correlation series r or ρ are interpreted as indicators of high similarity, the threshold is defined as:

$$\theta_S = \mu(S) + \lambda\sigma(S) \quad , \tag{4.22}$$

where S is substituted for the set of all values V of a correlation series with $f = r$ or $f = \rho$. The corresponding set of above-threshold values A is defined as a subset of V:

$$V(\mathcal{X}, \mathcal{Y}, l_p) = \{f(\mathcal{X}, \mathcal{Y}, n, l_p) \,|\, 1 \leq n \leq N - l_p + 1\} \tag{4.23}$$

$$A(\mathcal{X}, \mathcal{Y}, l_p) = \{v \in V(\mathcal{X}, \mathcal{Y}, l_p) \,|\, v > \theta_V\} \tag{4.24}$$

A boundariness function for a probe $\mathcal{X}$ in relation to a memory sequence $\mathcal{Y}$ can now be defined as:

$$m(\mathcal{X}, \mathcal{Y}, l_p) = \mu(A(\mathcal{X}, \mathcal{Y}, l_p)) \tag{4.25}$$

$$m'(\mathcal{X}, \mathcal{Y}, l_p) = \mu(A(\mathcal{X}, \mathcal{Y}, l_p)) - \mu(A(\mathcal{X}, \mathcal{Y}, l_p - 1)) \qquad (\text{with} \quad 1 \leq l_p \leq T_p) \tag{4.26}$$

The two boundariness functions m and m' are computed in the experiment in function `getHits` in `getSimilarity` (cf. listing B.29). The above-threshold values are referred to as "hits" in the source code. After all similarity measures have been computed, `experiment1` stores the results and produces graphs, according to the setup (cf. source code listings B.30 and B.31 on pages 314 and 319, respectively).

Results

For each run of the experiment, a new probe and a new memory sequence is constructed. This section presents a number of representative experimental results for constant shift corpora. The total length of the probe is set to $T_p = 100$ in experimental runs for which results are reported here. As the overall results of the 'whole-sequence' segmentation experiments are essentially negative, no extensive evaluation for all possible values has been performed (see section 4.4.5 for a general discussion).

The basic scheme of the following figures is the same for all results: The horizontal axis correspond to the probe length l_p. The vertical axis corresponds to the value of the respective similarity functions. Vertical dotted lines indicate true segment boundaries in the memory sequence $\mathcal{Y}$ aligned with the probe $\mathcal{X}$.[27] The label "X" marks the target segment and the label "..." marks the random parts in the corpus.

Figure 4.5 on page 144 shows the global similarity score C and its derivative C' based on normalised correlation scores ($f = \rho$) for four different parameter combinations. The first two rows (a) and (b) show results for some arbitrary settings of l_p and N. A mel-frequency cepstral coefficient (MFCC) representation with an effective frame rate of 500 Hz is used to encode the speech data that is used later in experiment 3 (section 4.4.4). In analogy to such a representation, a target length of $l_t = 17$ in this experiment can be interpreted as corresponding to 34 ms, assuming an equivalent frame rate for the artificial data. The parameters for the third and fourth row in figure 4.5 are examples taken from speech data. The target length $l_t = 29$ corresponds to the mean length of a [d] segment in the data from the IMS Corpus (cf. section A.2). The shift $l_\Delta = 2039$ corresponds to the respective mean distance between two occurrences of a [d] segment. Accordingly, the target length $l_t = 32$ corresponds to the mean length of a [ɛ] segment in the data taken from the Polish BOSS corpus (cf. section A.3), and the

[27] Note that in experiments 1 and 2 the material in the probe in the range $l_t < t_i \leq T_p$ consists of additional random frames and does not match the aligned corpus data.

shift value $l_\Delta = 535$ to its mean distance. The C graphs in (b) and (c) in figure 4.5 show a local maximum in C at the exact length of the target segment. The corresponding C' graphs show a local minimum at that index. There is, however, no clear trend visible that would indicate that this is the usual outcome. The first and the fourth row in figure 4.5 are two representative examples.

Figure 4.6 on page 145 shows the boundariness functions m and m' as well as the standard deviation σ of the corresponding sets of above-threshold values in the correlation series. The left panels show a local maximum in m at the exact length of the target segment in figure 4.6 (a) and (b). A corresponding local minimum is visible in m' in the middle panels. The boundary peak in m is flattened with increasing shift lengths to an extent where it is no longer visible, as the left panels in figure 4.6 (c) and (d) show. The corresponding minima in m' are hardly recognisable. The standard deviation $\sigma(A(\mathcal{X}, \mathcal{Y}, l_p))$ of the above-threshold values appears to be a more promising measure for boundariness. It often shows a peak at the true segment length as can be seen in the right panels of figure 4.6. The graph shown in 4.6 (b), however, demonstrates that this measure is not reliable in all cases.

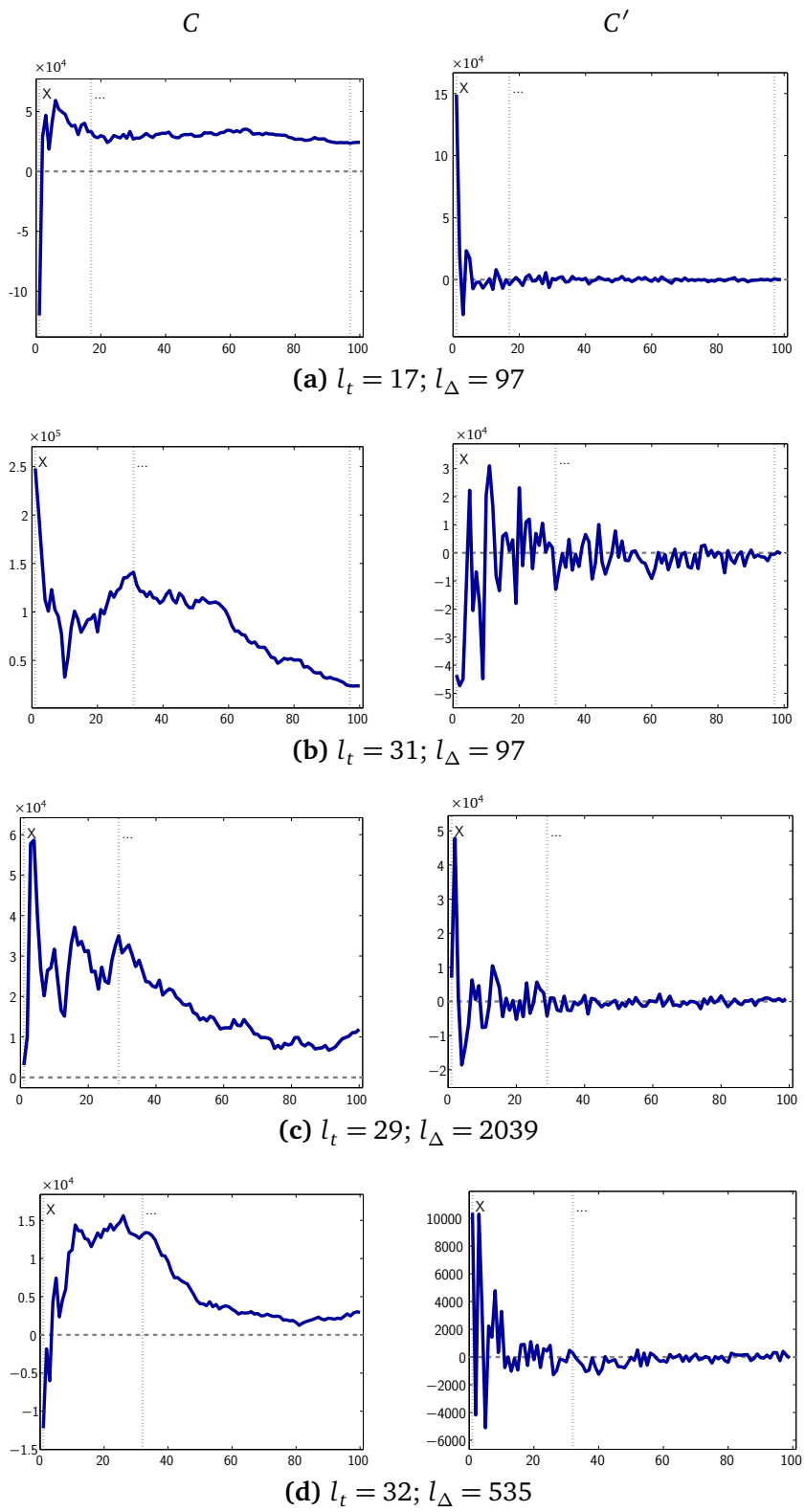
Note that the results shown in figure 4.6 on page 145 are based on the same experimental runs as the results shown in figure 4.5 on page 144. These examples illustrate how the functions m, m' and σ provide better indicators for the segment boundary than the functions C and C' on the same probe $\mathcal{X}$ and memory $\mathcal{Y}$.

4.4.2. Experiment 2: Variable-shift artificial data

The second series of experiments used *variable-shift corpora* which were constructed in the same way as the constant-shift corpora with the exception that the shift parameter l_Δ was not constant. Instead, the shift was variable. Random values from the interval $[l_p + 1, \alpha \cdot l_p]$ were taken for each successive occurrence of the target segment in the corpus. This particular interval was chosen in order to avoid overlap of the target segments, immediate repetitions and too long distances between two instances of the target segment. Such variable-shift corpora are slightly more realistic than the constant-shift type since the occurrences of a target segment are not evenly distributed across a corpus of natural speech. However, the task of finding the target segment in the probe based on such corpora is still expected to be easier than on actual speech data.

Method and implementation

The implementation of this experiment is based on experiment 1 with constant-shift corpora. The main MATLAB script for this experiment is `runExperiment2`. The commented source code is shown in listing B.32 on page 321. As in experiment 1, the main script initialises the experimental parameters and starts the actual experiment function — in



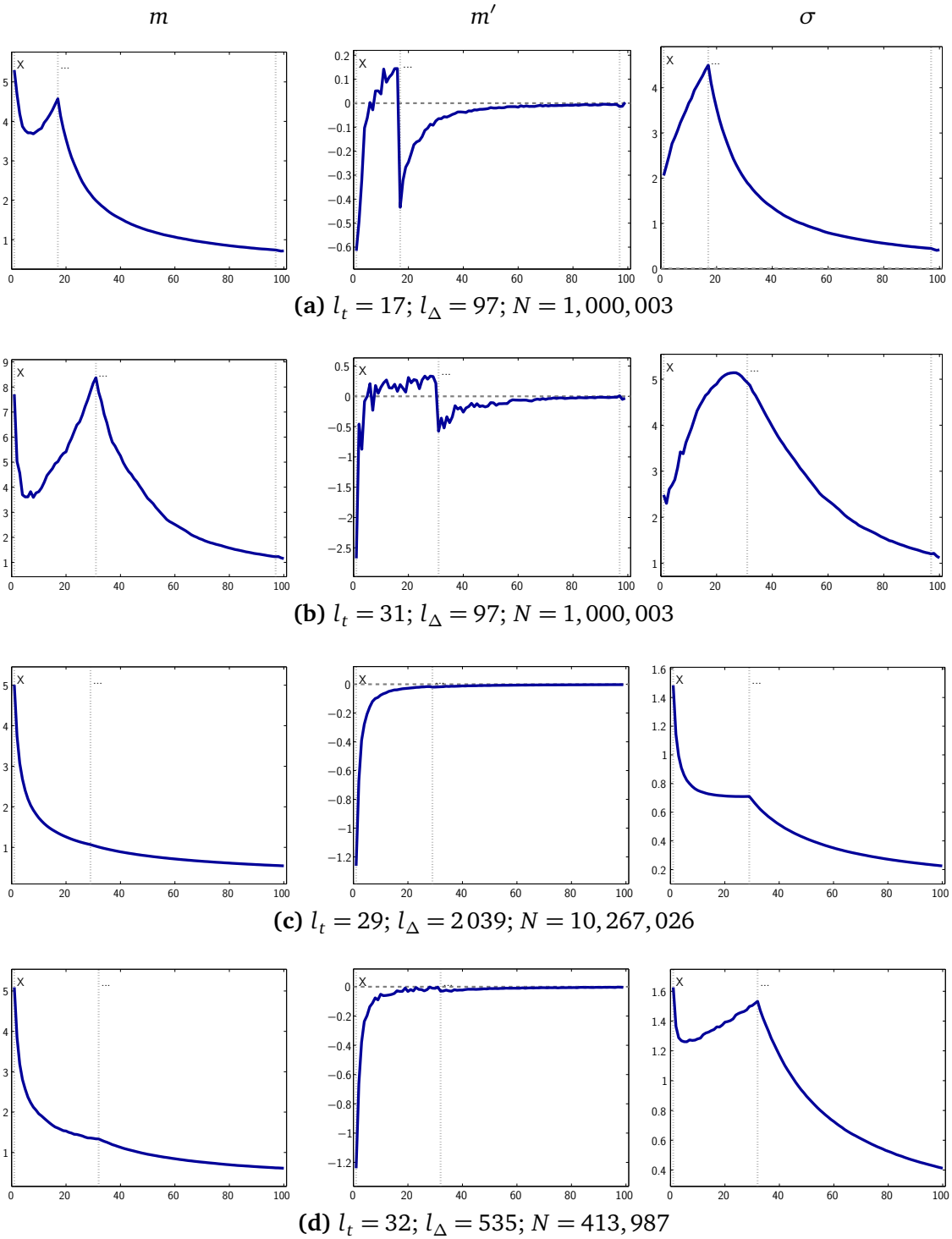
(a) $l_t = 17$; $l_\Delta = 97$

(b) $l_t = 31$; $l_\Delta = 97$

(c) $l_t = 29$; $l_\Delta = 2039$

(d) $l_t = 32$; $l_\Delta = 535$

Figure 4.5.: Similarity scores C and C' on constant-shift corpora with $f = \rho$. Horizontal axes in all panels correspond to the probe length l_p.



(a) $l_t = 17$; $l_\Delta = 97$; $N = 1,000,003$

(b) $l_t = 31$; $l_\Delta = 97$; $N = 1,000,003$

(c) $l_t = 29$; $l_\Delta = 2\,039$; $N = 10,267,026$

(d) $l_t = 32$; $l_\Delta = 535$; $N = 413,987$

Figure 4.6.: Boundariness on constant-shift corpora with $f = \rho$. Left column: m; middle column: m'; right column: σ. Horizontal axes in all panels correspond to the probe length l_p. See text for further details.

this case `experiment2`, shown in listing B.33 on page 323. The only relevant difference between experiment 1 and experiment 2 is the way the memory sequence $\mathcal{Y}$ is constructed. Function `createVariableShiftMS` generates an artificial corpus with a repeated target segment embedded into a sequence of random vectors according to the given parameters. The source code of this function is listed in B.34 on page 324. To create the random shift values, MATLAB's random number generator `randi` is used. This generates uniformly distributed integers within a defined range. The remaining code is shared with experiment 1 and not repeated here. These are the MATLAB functions for the creation of a probe, the computation of the similarity functions and the production of the result graphs (for details, see the previous section and appendix B.1.2).

Results

Figure 4.7 on page 147 shows the results of the boundariness functions m, m' and σ for four parameter combinations. The target length parameters l_t are chosen according to the results shown for the constant-shift corpora in order to allow comparisons between the two approaches. For the results shown in figure 4.7 (c) and (d), the parameter α was chosen such that the mean shift l_Δ corresponds approximately to the values for the constant-shift corpora and thus to the underlying values from the speech corpora.

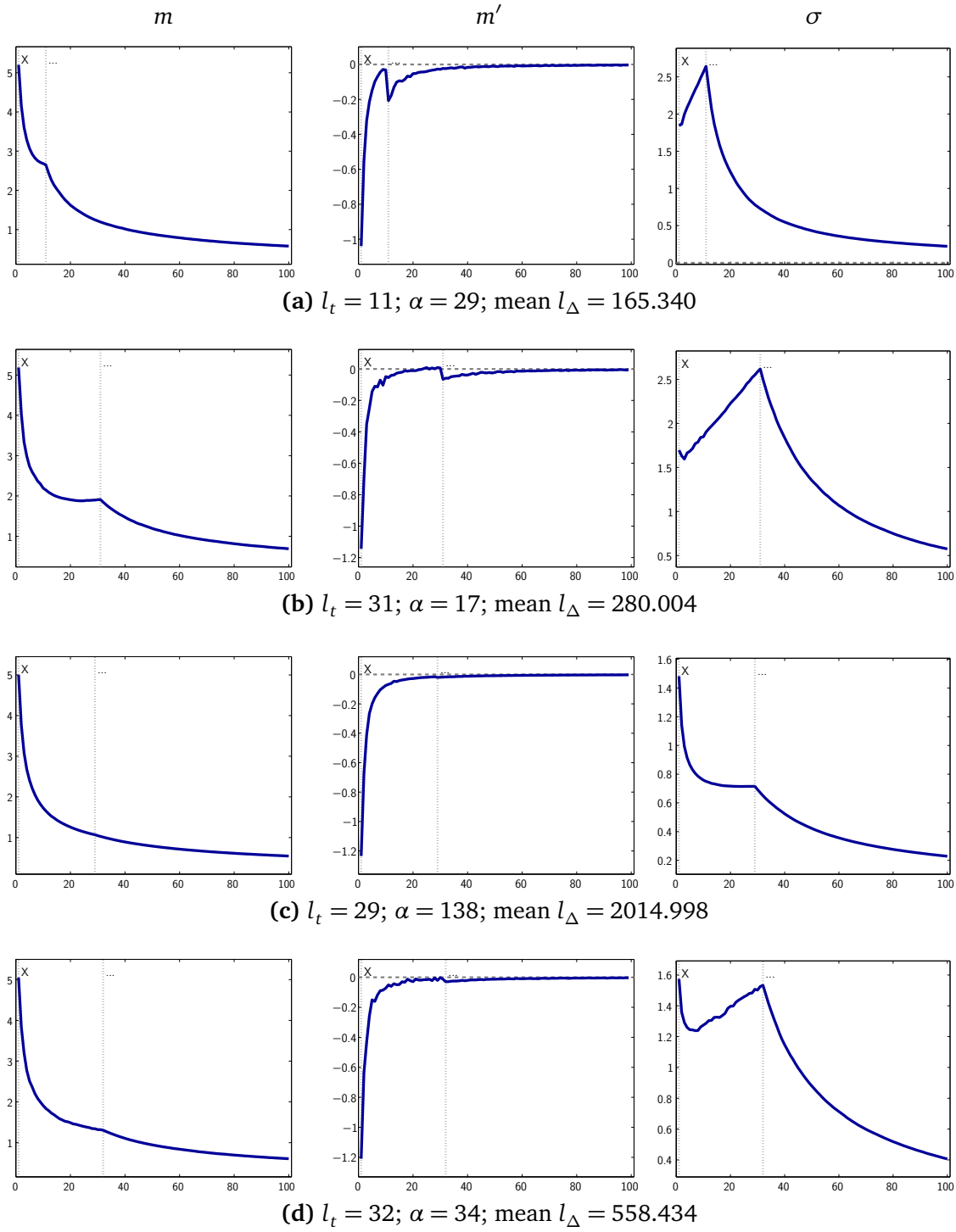
The graphs for the first two results in 4.7 (a) and (b) show that all three functions m, m' and σ reflect the true segment length l_t by either a local maximum or a local minimum at $l_p = l_t$. In case of m, the global maximum corresponds to the beginning of the probe. Accordingly, the global minimum in m' corresponds to the beginning of the probe. This is a trivial boundary which could easily be ignored by an automatic peak-picking algorithm.

In figure 4.7 (c) and (d) the same picture as in experiment 1 can be observed on the third and fourth results: The peaks in m are vanishingly flattened by the larger values of l_Δ, which correspond to real-speech examples. Again, the standard deviation σ on the correlation values above the defined threshold provides a better indicator for the true segment length in the probe.

4.4.3. Baseline results

The approaches from experiments 1 and 2 have been applied to non-target probes, i.e. probes which do not contain the target segment. Random probes are generated according to the same scheme as the generation of the artificial corpora. Running the segmentation experiments on such a non-target probe is triggered by an input parameter of the MATLAB functions — see the source code listings in appendix B.1.2.

The results on non-target probes always look similar to the ones shown in figure 4.8: There seems to be no correlation between the boundariness function values and the



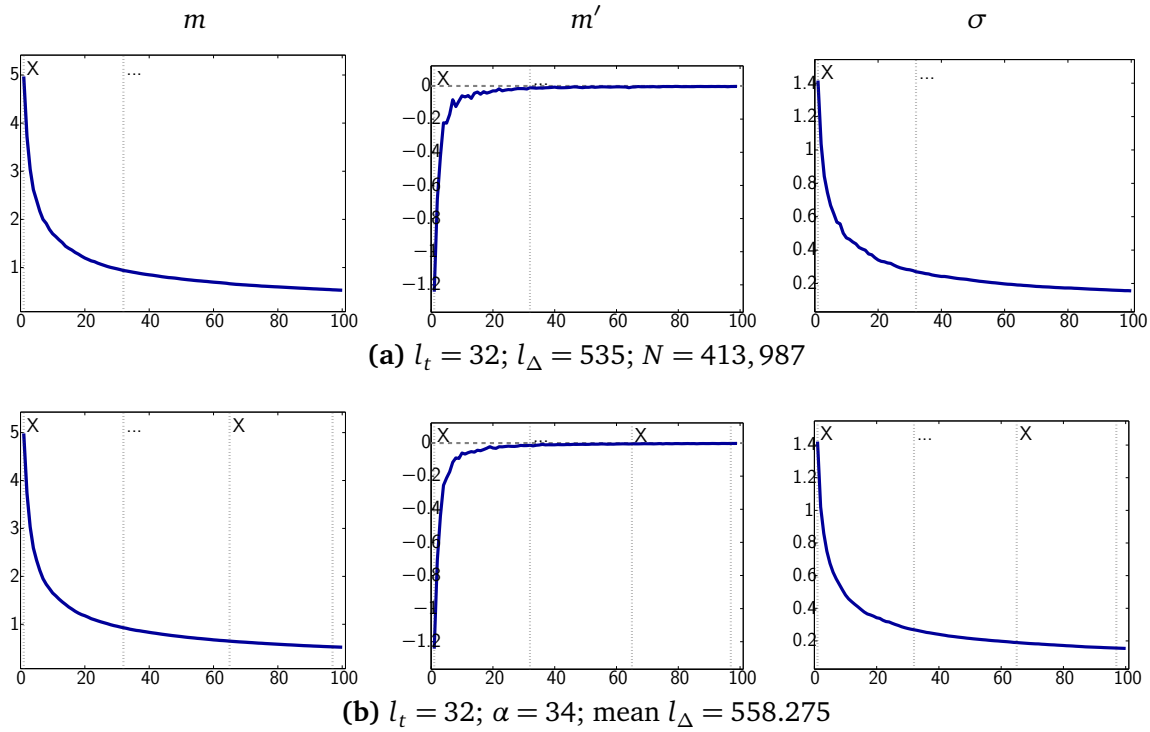
(a) $l_t = 11$; $\alpha = 29$; mean $l_\Delta = 165.340$

(b) $l_t = 31$; $\alpha = 17$; mean $l_\Delta = 280.004$

(c) $l_t = 29$; $\alpha = 138$; mean $l_\Delta = 2014.998$

(d) $l_t = 32$; $\alpha = 34$; mean $l_\Delta = 558.434$

Figure 4.7.: Boundariness on variable-shift corpora with $f = \rho$. Left column: m; middle column: m'; right column: σ. Horizontal axes in all panels correspond to the probe length l_p. See text for further details.



(a) $l_t = 32$; $l_\Delta = 535$; $N = 413,987$

(b) $l_t = 32$; $\alpha = 34$; mean $l_\Delta = 558.275$

Figure 4.8.: Baseline results for segment boundariness experiments with $f = \rho$. Left column: m; middle column: m'; right column: σ. Horizontal axes in all panels correspond to the probe length l_p. See text for further details.

segmentation boundaries of the corpus. Thus, the approach does not produce false positive segmentation boundaries.

4.4.4. Experiment 3: Speech data

Finally, a third series of experiments on speech segmentation was carried out using speech data as input. The same approach of direct comparisons between a short probe and a long memory sequence is applied as in the two experiments on artificial data.

Data

The data for experiment 3 was taken from two speech corpora. The first one is the IMS unit selection corpus which contains recordings of a German professional male speaker (Schweitzer et al., 2006, see also appendix A.2). The corpus comprises 2776 individual files, each containing an utterance (usually not more than one or two sentences). The original data is sampled at 16,000 Hz. 12-dimensional MFCCs were computed using the `mfcc` function of the AUDITORY TOOLBOX (Slaney) for MATLAB. MFCCs are considered to

be a suitable approximation of the speech representation in human speech perception and they are applied in various simulation experiments (e.g. Lin, 2005, p.39) as well as in speech processing tasks like ASR (cf. for example Morgan et al., 2004). The parameter `frameRate` of the `mfcc` function was set to 500 (corresponding to a 2 ms window shift). All remaining parameters were not changed from their respective default values. The vectors were then length-normalised. The parameters for the segmentation experiments using speech data are taken from the phonetic annotations of the IMS corpus. In total, the corpus comprises 107,209 phone segments. In total, the entire processed IMS corpus for this experiment is represented as a sequence of 4,310,124 frames, each of which is represented by an MFCC vector.

In addition to first results reported by Duran et al. (2010b), I present here results based on speech data taken from a Polish corpus. The material is taken from one part of the Polish BOSS corpus (see appendix A.3 for details). The same pre-processing is applied to the Polish corpus data such that it is encoded as a 12-dimensional sequence of MFCC vectors with an effective frame rate of 500 Hz. In total, the part of the Polish corpus used for this experiment contains 11,662 phone segments, parametrised as a total of 413,987 frames.

Method and implementation

The implementation follows the same scheme as the one used in experiment 1 and 2 with artificial data. The MATLAB script `runExperiment3` is used to initialise the experimental parameters and the environment, the commented source code of which is listed in B.35 on page 326 in the appendix. The actual segmentation experiment is implemented in function `experiment3`, shown in listing B.36 on page 327.

The memory sequence is initialised from a speech corpus which has been prepared as described above and stored in a binary MATLAB file format. The probe is defined as a stretch of speech of length T_p taken from the memory sequence. The probes are selected such that they are taken from the onset of an utterance or that they are at least preceded by silence. Silence here is defined as either any segment of the corpus explicitly labelled as such or any non-labelled part of the recordings. This is motivated by the assumption that the language learner makes use of a separate processing module to distinguish speech from non-speech. In that sense, the approach resembles the *utterance-boundary strategy* to speech segmentation (discussed in section 4.1.1). The implementation of `experiment3` only requires that a specific segment is specified at the beginning of which the probe is supposed to begin. The experimental parameters are selected manually such that the probes correspond to the beginning of an utterance with a total probe length of $T_p = 100$. The remaining MATLAB code is shared with experiments 1 and 2, and can be seen in section B.1.2 of the appendix.

Results

Some representative results for the German and Polish speech corpora are shown in the following figures. Figure 4.9 on page 151 shows results for two runs of the experiment on the German data. The top row shows the results for a probe taken from an utterance beginning with [das]. The probe length is again set to $T_p = 100$ frames, as in the previous experiments. A local minimum seems to coincide with the first segmental boundary between [d] and [a]. As [d] appears frequently in the corpus[28], the comparison procedure might find sufficient instances of similar segments in the corpus to identify the presence of a segmental boundary. The second row in figure 4.9 shows the results for a probe taken from an utterance beginning with [mat]. Overall, [m]-segments are less frequent in the corpus[29], but the procedure still appears to capture some of the boundariness information for the first segment in the [mat]-probe. As for the first example, a local minimum in the boundariness functions occurs at (or close to) the first segmentation boundary in the probe.

Figure 4.10 on page 152 shows results for two runs of the experiment on the Polish data. The overall picture is the same. The top row in figure 4.10 shows an example with a probe starting with [tɔ#na], where # indicates a word boundary. A weak indication of the first segment boundary is visible between the [t] and the [ɔ] segment. Additionally, a peak in m' at the boundary between [ɔ] and [n] might be interpreted as an indication of the present word boundary. The bottom row in figure 4.10 shows the results for a probe starting with [mi#fa]. Here, in contrast, only one potential boundary is visible in the boundariness graphs.

The examples presented here indicate that the whole-sequence approach might be useful in some cases. Overall, however, the results are not very promising. Although moderate results can be achieved on the artificial data, the approach fails on real speech.

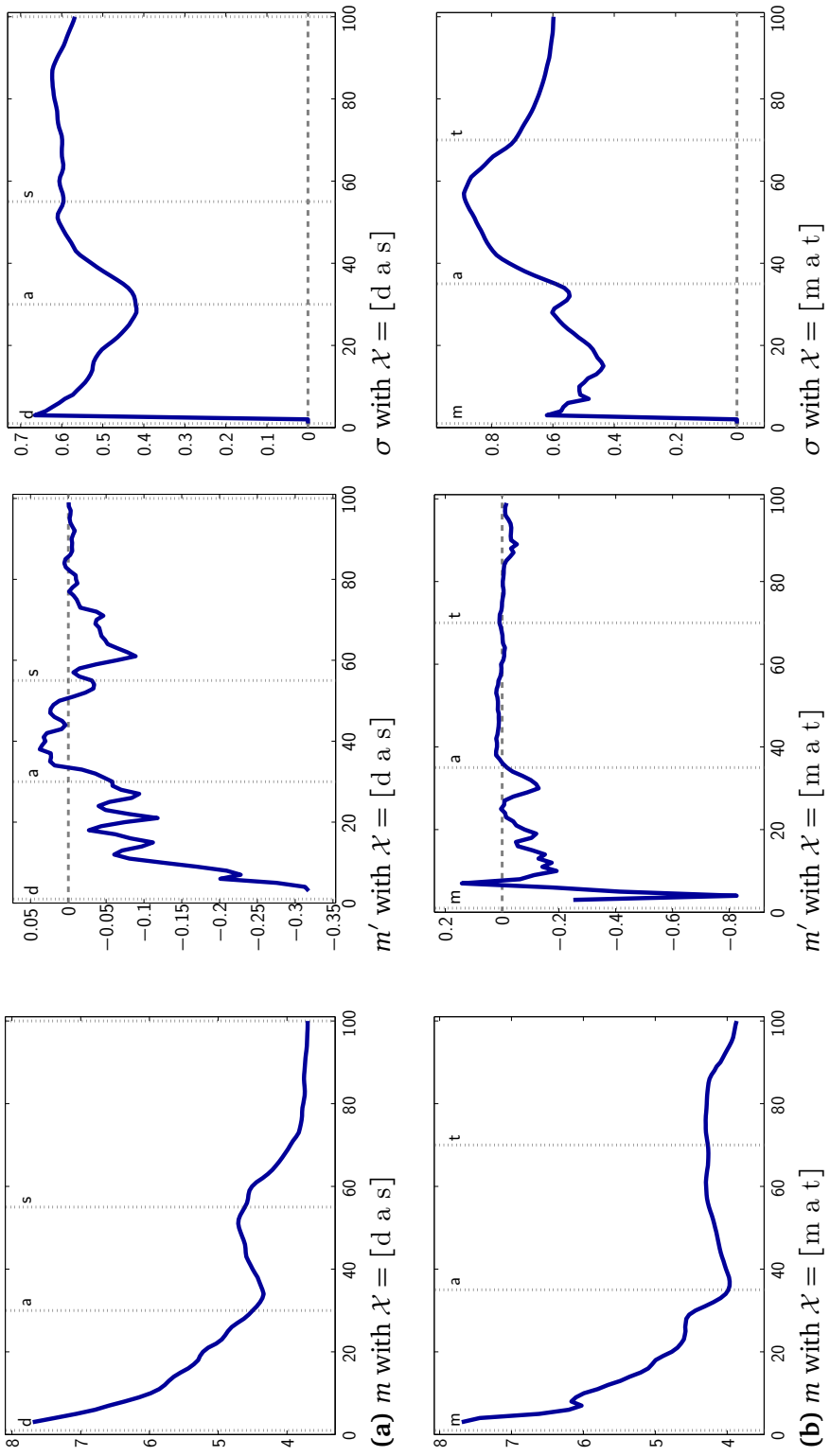
4.4.5. Discussion of the whole-sequence approach to segmentation

The experimental results of the whole-sequence segmentation experiments are essentially negative. Wade et al. (2010, p.237) speculated that a useful speech segmentation method might follow straightforwardly from direct signal comparisons of a short stretch of speech (a probe) and a long sequential speech memory. The results of the experiments presented in this section demonstrate that this is probably not the case.

There are some aspects of the whole-sequence segmentation approach that require further attention. The artificial corpora used in experiments 1 and 2 of this section provide useful input for the simulation with known properties. Employing artificially

[28] In the IMS corpus as used for this experiment there are in total 5,035 [d]-segments, 1,421 of which appear as the first segment in an utterance (ignoring labelled silence segments).

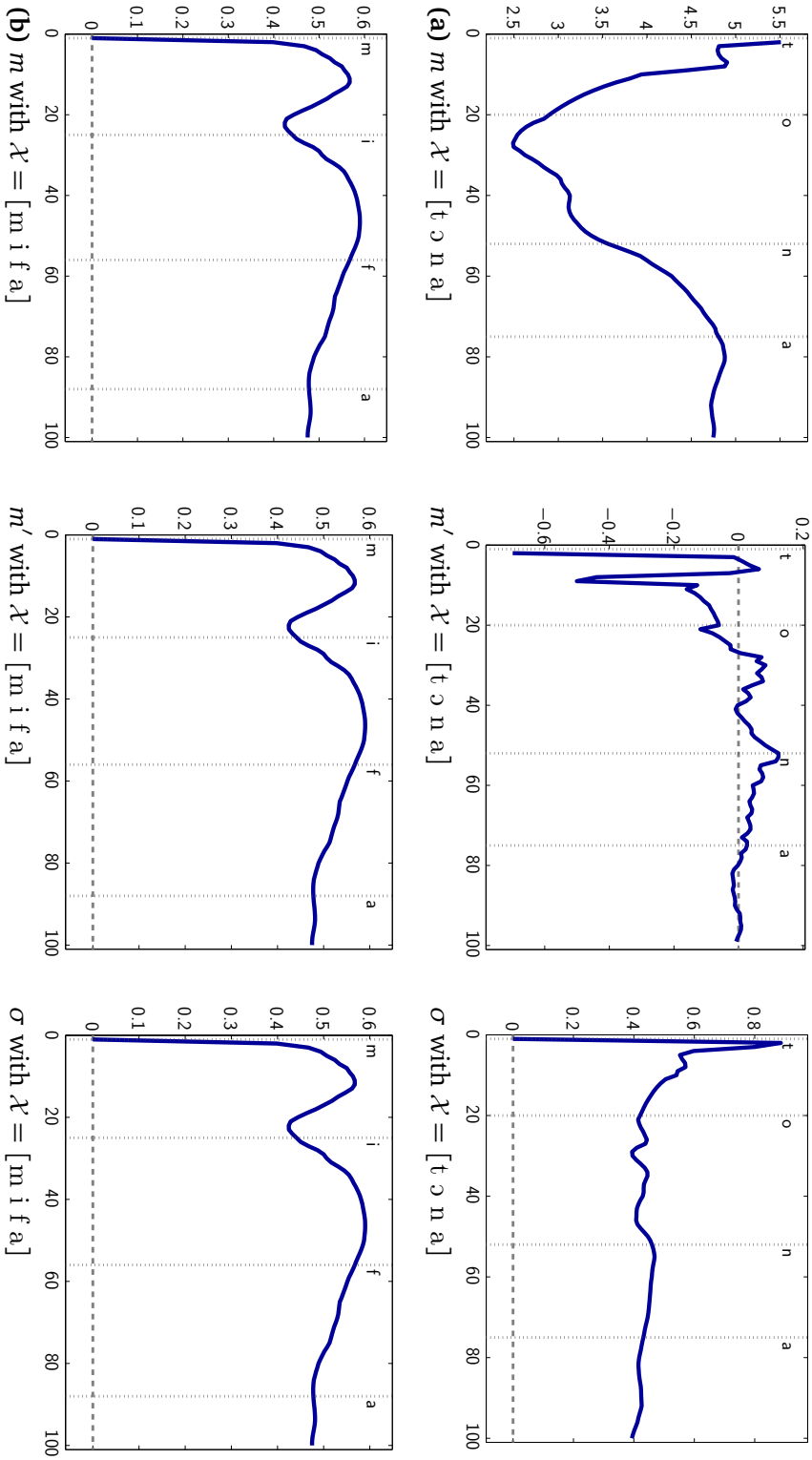
[29] There are in total 3,575 [m]-segments, 374 of which are utterance initial.



(a) m with $\mathcal{X} = [\mathrm{d\ a\ s}]$ m' with $\mathcal{X} = [\mathrm{d\ a\ s}]$ σ with $\mathcal{X} = [\mathrm{d\ a\ s}]$

(b) m with $\mathcal{X} = [\mathrm{m\ a\ t}]$ m' with $\mathcal{X} = [\mathrm{m\ a\ t}]$ σ with $\mathcal{X} = [\mathrm{m\ a\ t}]$

Figure 4.9.: Results of Segmentation experiment 3 on IMS unit selection speech corpus.



(a) m with $\mathcal{X} = [\mathrm{t\ ɔ\ n\ a}]$ m' with $\mathcal{X} = [\mathrm{t\ ɔ\ n\ a}]$ σ with $\mathcal{X} = [\mathrm{t\ ɔ\ n\ a}]$

(b) m with $\mathcal{X} = [\mathrm{m\ i\ f\ a}]$ m' with $\mathcal{X} = [\mathrm{m\ i\ f\ a}]$ σ with $\mathcal{X} = [\mathrm{m\ i\ f\ a}]$

Figure 4.10.: Results of Segmentation experiment 3 on Polish BOSS speech corpus.

constructed data as an approximation to a speech signal — with various degrees of abstraction — is an often applied technique in computer simulation studies on speech (cf. for example Wade et al., 2010, p.234; Mitra et al., 2012, p.2273). However, the generation method applied here is very simple. The fact that the results on artificial data do not carry over to real speech data suggests that there are some important properties of acoustic speech signals missing in the artificial data. A statistically more speech-like artificial memory sequence could be generated as, for example, suggested by Gersho and Gray (1992, p.382). They use "well-defined random processes that exhibit similarity with speech" to examine their vector quantization method, and they point out that "this is of interest because it might both suggest performance bounds and indirectly shed light on how good such models of real data sources are." The method of generating the artificial data in experiments 1 and 2 of this section is based on a pseudo-random number generator based on a normal distribution. The dimensionality of 12 was chosen according to the 12-dimensional MFCC parametrisation of the speech data used in experiment 3. Additionally, several settings of the parameters for the target segment length and the total length of the memory sequence have been tested that were drawn from the speech corpora. Despite the simple generation of a sequence of random vectors, the artificial corpora can be considered useful approximations to the acoustic speech signal for the purpose of the experiments.

The probes in experiments 1 and 2 are created by appending random vectors to the target segment. In experiment 3 the probes are taken entirely from the memory sequence. This approach should also be tested with the artificial data. Moreover, the artificial data could be constructed from pseudo-segments such that there are multiple repeated patterns of different types in the corpus in addition to the target segment. The influence of the size of the segment type inventory could be tested with such an approach as well as the influence of various distribution patterns.

Based on observations such as those reported here for the whole-sequence segmentation approach, we (i.e. Duran et al., 2010b, p.153) hypothesised that successful segmentation is only possible if intervals of speech are considered *in context*. The direct signal comparisons presented here do not consider the context of the frames from the probe or the memory sequence. Speech is highly variable. At the level of individual frames of 2 ms length, there can be very similar instances which belong to different phonetic categories on the one hand, and there can be very dissimilar instances which belong to the same phonetic category, on the other. We (Duran et al., 2010b, p.157) proposed an *extended context method* to incorporate context information into the segmentation process. Using context information for segmentation faces a chicken-and-egg problem. The segmentation decision at any given point t in time depends on its preceding and its following context. These contexts are potentially very long, and the further away any part of the context is from t the less important it will be for the segmentation at t. Thus, it appears, the context needs to be segmented into meaningful units such

that the information taken into account can be restricted to the parts of the context immediately surrounding t. The definition of such a segmentation of the context, in turn, depends on the decision at time t. In order to avoid this circularity, we introduced the extended context method which is based on *context frames*. For a given frame f_t at time t, its left and right context frames are defined as the first frames which differ from f_t by more than a set threshold. The underlying assumption is that the segmentation procedure must be sensitive to acoustic changes. This way, longer steady state intervals will be skipped, while immediately neighbouring frames will be considered in regions of great acoustic changes. Simple segmentation methods were defined based on a number of boundariness functions derived from the corpus data: the Euclidean distance between adjacent frames, the minimum pair distance within a fixed context window, the transition probability between frames and the entropy. These functions were learned on one part of the corpus and the other part was left for evaluation. A binary segmentation function was defined on the continuous boundariness values over the memory sequence (Duran et al., 2010b, p.147). Statistical methods such as these fail necessarily on real speech data due to the fact that there are no two identical stretches of speech at the acoustic level. This means that any pair of acoustic frames is probably unique. To address this problem, a *clustering* methodology was used to group similar segments. For this purpose a vector quantization (Gersho and Gray, 1992) was computed from the MFCC representation of the corpus using *bisecting k-means* (Steinbach et al., 2000a)[30] and Euclidean distance. 1,000 clusters were generated which reflect dense regions in the MFCC space. Replacing the 12-dimensional MFCC vectors by 36-dimensional extended context vectors (12 MFCC components from the current frame and the left and right context frames, respectively) proved a much better basis for segmentation decisions than relying on immediately neighbouring frames (Duran et al., 2010b, p.159).

Another problematic aspect of the experiments presented here that requires further investigation is the type of data that was used. Restricting the simulations to acoustic sounds was actually motivated by practical considerations. The model of "correspondence learning" as proposed by (Duran et al., 2010b, p.135) would also require the consideration of articulatory information in a complete account of the acquisition problem. The segmentation method needs to be applied to corpora that contain both acoustic and articulatory data. Multiple layers or parallel streams could be defined which provide multiple sources of information and cues for segmentation decisions. The boundariness graphs for the experiments with real speech data resemble the activation graphs presented by Johnson (1997b, p.107f). The evidence for segmental boundaries appears to be subtle and it may require incorporation of cues from various sources.

[30] See also section 5.3.2 on page 183, where the bisecting k-means clustering method is discussed in more detail.

4.5. Discussion on boundariness and segmentation

This chapter addressed the problem of speech segmentation and the notion of gradual *boundariness*. Segmentation of a speech signal is not only crucial for applications like automatic speech recognition. It is also of crucial importance for computational models of speech perception or language acquisition. From a psycho-linguistic and cognitive perspective, simulations are interesting tools to investigate how segmentation can be learned from the speech stream and whether infants are ideal learners (Goldwater et al., 2009, p.40). From a phonetic point of view, simulations of human speech segmentation can help to investigate what sources of information are available in the speech signal to cue segment boundaries. The purpose of simulation experiments on speech segmentation is not only to investigate the production of linguistically plausible segmentations of a speech signal but also plausible segmentation methods and plausible mechanisms for learning them. Segmentation in speech technology often aims at achieving the first of these goals: producing a plausible segmentation. However, some methods don't even try to do that, but instead leave the problem of finding plausible segments to a post-processing module that determines the most likely sequence of segments. Maybe, human speech segmentation has more in common with speech technology. Maybe there is an initial segmentation of the speech stream that does not correspond to traditional phonetic or phonological units. Phonetic segments might then be described as an emergent phenomenon based on pre-linguistic auditory analyses.

It has been argued that perceptual speech units must be distinguished from (lower level) auditory units on the one hand, as well as from (higher level) phonological units on the other (e.g. by Massaro, 1996, p.87f). This raises the question whether phones are adequate units of phonetic analysis if they do not correspond to the actual units of perception. Most speech segmentation models assume a particular type of target units (e.g. words) and attempt at discovering these units in the speech stream. Often, a fundamental unit is presupposed by a specific model. This might appear as circular reasoning (especially in first language acquisition). The process of segmenting the speech stream at a particular level of linguistic abstraction, depends on the segmentation at another level of abstraction. This problem was discussed in the context of the *symbolic-sequence approaches*. Some simulation studies seek to address this issue by operating directly on quasi-continuous representations of speech. My experiments presented in section 4.4 also fall into this category. Continuous input data introduces difficulties which symbolic-sequence approaches do not face. In particular, there is no inventory of discrete fundamental units that can be subjected to identity comparisons. Some researchers might argue that there is actually no context- and speaker-independent fundamental unit of speech or language as discussed in section 4.1. In contrast to simple information processing based on bits of information, language processing involves "compositional signals in which the unit of interest is above the level of an indivisible bit" (Wedel, 2009, p.2).

The contrast between the apparent discreteness in speech perception, based on the ability of abstraction, and the continuous nature of speech needs to be investigated. The fact that it is a hard task for humans to manually segment speech data on the phonetic/phonemic level (Wesenick and Kipp, 1996) deserves much more attention. Trying to simulate a problem solving process that might, in fact, be performed by humans radically different from the educated linguists' intuitions could lead research to a wrong direction.

The question might arise, why are all these psycholinguistic studies discussed in section 4.1 relevant for phonetics and phonology? The experiments described in this chapter address the question whether phones and phonemes — the traditional units in phonetics and phonology — are really fundamental units of speech. At least, the question has to be asked, whether phones are actually useful units in first language acquisition. Research in phonetics and phonology needs to incorporate ideas from related disciplines — both theoretical and methodological — like psycholinguistics, speech technology or computational linguistics.

The experiments presented in sections 4.3 and 4.4 both make use of artificial data to investigate the behaviour of an algorithm under controlled conditions. Gersho and Gray (1992, p.382) use "simulated speech" to evaluate the performance of their vector quantisation method. They argue that it is not possible to evaluate the procedure against theoretical optima for speech data, "because no such optima are known." The approach taken in section 4.3 to use diagnostic segmentation methods is in part motivated by the fact, that a theoretical analysis of the evaluation measures is not sufficient, because the properties of the segmented speech data are not known to an extent that allows a rigorous formal evaluation.

5. Unsupervised classification of articulatory and acoustic speech recordings

The previous chapter focused on speech segmentation and addressed the question of how a continuous, acoustic speech signal can be segmented into linguistically appropriate units. This chapter presents a series of clustering experiments which were performed on a corpus of speech and electromagnetic midsagittal articulography (EMA) data. The goal of these experiments was not primarily to investigate the absolute quality of a particular clustering method, but to compare the clustering results on three different types of data: (1) articulatory data as represented by EMA measurements, (2) acoustic speech recordings and (3) a combination of these two.

The experiments presented in this chapter relate to two research fields. The first two sections of this chapter give an overview of related work. First, related work is briefly discussed in section 5.1 where articulographic recordings are used in computational studies or simulation experiments. Second, the method of data clustering is discussed in section 5.2. After a general overview of different clustering methods and ways to evaluate the results, some examples of computational studies from phonetics and phonology are presented, where clustering approaches have been employed. The experiments presented in the following two sections of this chapter investigate EMA speech signals in comparison to acoustic speech signals in an unsupervised learning scenario under two different linguistic research questions. A first series of clustering experiments is presented in 5.3. 5.4 presents an experiment on Polish vowel classification in different phonological contexts. A general discussion of the presented experiments and the clustering methodology concludes this chapter.

5.1. Articulatory and acoustic speech data in computational studies and simulation experiments

Obtaining recordings of articulatory gestures and vocal tract shapes during speech production is a laborious and time consuming process — especially if compared to the process of obtaining acoustic speech recordings. Still, such data can be useful in the

study of a number of different research questions and it has been used in simulation experiments and computational studies on human speech. Articulatory measurements are, for example, used by Bruni (2011, p.111ff) to study gestural coordination in consonant clusters. For practical application purposes, articulatory measurements have been used in the development of articulatory speech synthesis (cf. Kröger and Birkholz, 2009, p.307) or for the improvement of automatic speech recognition (ASR) (cf. Wrench, 2000, p.1f). The expectation in research and development in the domain of speech technology is often that the articulatory data contains more or complementary information as compared to the acoustic speech signal.

A study in the domain of ASR by Silva et al. (2007) investigates the usefulness of articulatory measurements in terms of their correlation with the phone segment labels (p.457). They analyse EMA measurement data of spontaneous speech recordings of American English and compare it to the corresponding acoustic data. They use k-means vector quantisation for the observation vectors and determine the amount of discrimination information by computing mutual information between the articulatory observation vectors and the corresponding discrete phone label vectors as a fidelity criterion (p.458f). Different representation schemes for articulographic signal recordings are examined and compared to each other. Silva et al. (2007, p.459) find "that the oral production signals from EMA carry valuable phone discrimination information albeit not to the extent of the acoustic feature vector." However, they also point out that their frame-by-frame method is biased towards the acoustic signal and that a representation considering the "long-term dependencies of the articulators [...] would be more relevant for the phone classification task" (p.460).

Another study in the domain of ASR is reported by Mitra et al. (2012). They carried out experiments on the incorporation of articulatory gestures (based on *Articulatory Phonology*) in a speech recognition system. They trained different ANN models to learn the mapping from acoustic to articulatory data. They conclude that the incorporation of articulatory information improves speech recognition accuracy over state-of-the-art acoustic speech representations used in ASR (p.2284).

A number of simulation experiments have been developed which implement a speech production–perception loop that incorporates both acoustic and articulatory data (e.g. Guenther et al., 2006; or Kröger et al., 2010). Howard and Messum (2011), for example, present a model which simulates infant speech acquisition which incorporates an articulatory synthesizer. The model uses feedback signals to guide its learning process. These signals include acoustic and articulatory (i.e. somatosensory) feedback (p.97). See section 2.4 for more information on speech perception/production models.

Articulatory data is used by Kello and Plaut (2004) to train a computational model of speech acquisition. The training is performed on real speech recordings taken from the MOCHA corpus which contains both acoustic and articulatory recordings[1]. The

[1] See section A.4 for more details on the MOCHA database.

connectionist model learns to produce an acoustic output based on a given articulatory input. An intelligibility test is performed as a measure of the amount of phonetic information captured by the model. Intelligibility of speech output produced by the simulation is measured by determining the accuracy of word recognition by eight human transcribers (native speakers of American English). The reported average percentage of correct words is around 60% on untrained tokens and around 80% on trained tokens. Kello and Plaut (2004, p.2362f) conclude "that the forward mapping from articulations to acoustics can be learned" and that it is "well-approximated" by their model.

Looking at the articulatory dimension can be justified from a theoretical point of view, considering that articulatory or gestural information has been suggested as constituting the primary representation of speech (units and organisation). It has been argued, for example, that speech units are stored as articulatory patterns and invariant motor commands which provide a basis for speech perception and production (Liberman and Mattingly, 1985, p.2f). In *Articulatory Phonology*, articulatory gestures are assumed to represent the basic phonological units (Browman and Goldstein, 1992).

Thus, taking into account ideas of a speech perception–production loop which integrates feedback from different modalities it seems reasonable to incorporate both acoustic and articulatory data in simulation experiments on speech processing. Theoretical considerations suggest further investigations of articulatory data. In addition, research on speech recognition and processing indicates that both the articulatory and acoustic modality provide useful resources for the extraction of linguistic information. We (Duran et al., 2010b, p.164) have argued that articulation could provide an important constraint in speech segmentation. The experiments presented in this chapter employ articulatory data in an unsupervised learning scenario and compare the results against results obtained based on acoustic data.

5.2. Clustering

> "clustering is in the eye of the beholder"
> (Jain, 2010, p.663)

This section is about *data clustering* (also referred to as *cluster analysis*, or often simply as *clustering*) which is a machine learning method where the term "cluster" has a different meaning from that commonly used in phonetics and phonology. Clustering, in the context of this section, is a method of grouping individual data items such that the similarity between items within the same group is maximized and that it is minimized for items from different groups. This machine learning method is applied in various research disciplines and applications, such as computer vision, marketing, business administration or computational biology and also in a wide range of NLP tasks (see, for example, Manning and Schütze, 1999, Chapter 14; or Jain, 2010, p.653 for an overview).

According to my definitions in section 1.2, data clustering, on its own, is not a simulation technique. As an instance of unsupervised machine learning, however, it is relevant to the overarching topic of this thesis. Moreover, it has been used in several simulation studies in the field of phonetics and phonology. Clustering of acoustic data is often applied as a method of unsupervised learning in computer simulations of speech sound acquisition, for example.

This section is structured as follows: First, I give a brief introduction to data clustering and discuss some necessary terminology and provide formal definitions in 5.2.1 and 5.2.2. Then, clustering methods are discussed in 5.2.3 with a primary focus on the *k-means* algorithm. Part 5.2.4 of this section addresses the evaluation problem and presents some commonly used evaluation measures. Some examples of studies are given in 5.2.5 where a clustering method is used in computer experiments in phonetics or phonology. Finally, this section is concluded by some general remarks on data clustering in 5.2.6.

5.2.1. A short introduction to data clustering

Data clustering can be categorised as a method of unsupervised machine learning. It takes as input a set of objects and produces as output a set of groups, the *clusters*, of the given objects. Applying a data clustering procedure to a data set is based on the assumption that there are separable "*natural* groupings" of items within the given data set (Jain, 2010, p.652). Based on this premise, clustering procedures are devised such that they create a partition on a data set.

Often, there are certain expectations about the solution, when clustering is applied. When F_1–F_2 vowel formant data items are clustered, for example, the number of expected clusters corresponds to the number of vowel categories. An illustration of such a clustering is shown in figure 5.1 on page 162: The left panel shows a set of points in a two-dimensional space. The underlying data set consists of normalised measurements of formant values of German vowels. The centre panel shows an example for one possible result of a clustering applied to this data set[2]. Clustering was performed using the *k-means* method (cf. section 5.2.3 below) as implemented in R (The R Foundation for Statistical Computing, 2011) with the `kmeans` function of the STATS package. The number of clusters was set to $k = 15$ and the remaining parameters were not changed from their defaults. Different symbols and colours are used to mark the items' cluster membership. The right panel shows the corresponding true labels for each data item from this data set. The number of clusters $k = 15$ is in no way obvious from the raw data shown in the left panel of figure 5.1. It has been selected here based on a priori knowledge about the true number of clusters, i.e. the vowel categories, and the expectation that

[2] The original data is taken from Duran (2008, p.90f): the example shows the first and second formant of all German monophthongal vowels, spoken by three bilingual speakers ('B02', 'B03', 'B04'), normalised according to Lobanov (1971).

the clustering procedure should produce 15 clusters. Without this knowledge, other numbers of clusters might seem equally or even more plausible[3]. Although this is a much simplified example, it is often the case that there are no obvious or "natural" clusters which are immediately visible in a data set. Often, the data is represented in a high dimensional abstract feature space which is not accessible to human intuition and which cannot be visualised in a straightforward manner. Another often encountered problem are overlapping or non-convex clusters. And, in most cases, a data set can be clustered in various ways which might seem equally plausible if there is no reference data available against which to evaluate the quality of the generated partition.

5.2.2. Terminology and formal definition

The terms "cluster" and "clustering" represent technical terms with differing meanings in different fields of research, at least three of which are relevant for this thesis. This constitutes a considerable source of terminological confusion.

In phonetics and phonology, the term *cluster* usually refers to a sequence of consonants in one syllable. This has to be distinguished from the more abstract term cluster in the domain of data clustering where it refers to a sub-set of objects that belong to a data set.

Of particular relevance to this thesis is the specific sense of the term *clustering* as it can be found in psycholinguistic literature on speech segmentation and on recognition of speech items (words or syllables) in the continuous speech stream (cf. section 4.1 on page 84). Goodsitt et al. (1993), for example, investigate possible strategies of speech segmentation in first language acquisition, and they distinguish between "clustering strategies" and "bracketing strategies". They define *bracketing* as segmentation based on cues to the endpoints of individual speech items (e. g. prosodic cues; p. 230). They define *clustering*, on the other hand, as segmentation based on high(er) predictability within speech items. Bracketing, according to Goodsitt et al. (1993, p. 231), presupposes the existence of segment boundary cues, while clustering presupposes the existence of "some basic element of perception (syllable or phone)". That notion of clustering must be distinguished from the one used in the context of data clustering. In speech segmentation, as in phonetics and phonology, the subject matter of investigation is *sequential* data. In data clustering, the abstract representation of the data items are not ordered sequentially. Thus, in the scope of cluster analysis as discussed in this section, the term clustering refers to a method of assigning individual items to a specific set, a "cluster", based on their similarity or dissimilarity to other items. I.e. clustering, in that sense, is based on internal properties or features of the considered items, which need not be in any particular order. Categorisation usually builds on a given segmentation

[3] For the sake of simplicity, the vowel formant data is reduced to the 2-dimensional $F_1 \times F_2$ vowel space in this example. The purpose is the illustration of the clustering principle and not the achievement of an optimal clustering solution.

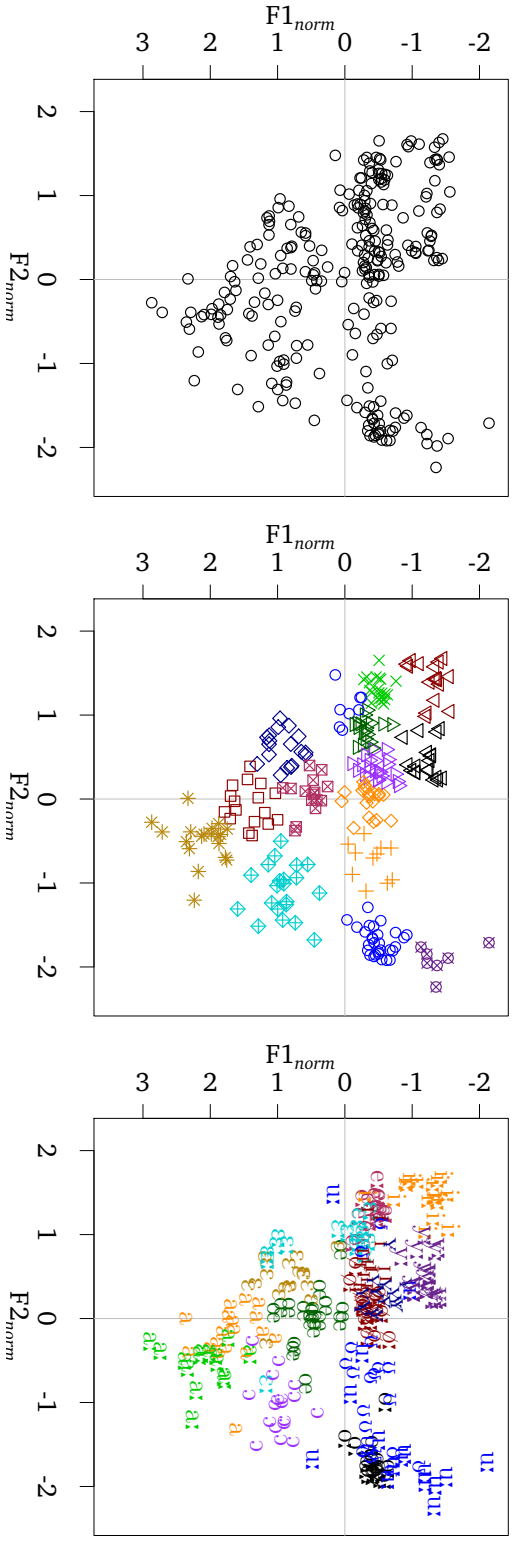


Figure 5.1.: k-means example: The unlabelled data set on the left, an exemplary result of a k-means clustering with $k = 15$ (the true number of clusters) in the middle, and the corresponding reference labels for each item in the right panel. This example illustrates how k-means generates contiguous clusters while the true clusters show considerable overlap. See text for details.

of the data, while the opposite is not necessarily true. It is important to distinguish these different meanings, especially when speech acquisition is considered. It can be argued that segmentation and categorisation of speech items are two related, yet distinct processes in language acquisition (cf. Duran et al., 2010b, p.137).

Terminology and notational conventions are adopted primarily from Manning and Schütze (1999); Manning et al. (2009) and Jain (2010). The following symbols and notations are used in this section to discuss data clustering methods and evaluation measures:

Notation	*Meaning*
$D = \{d_1, d_2, \ldots, d_N\}$	the data set of N items to which a cluster analysis is applied
$\Omega = \{\omega_1, \omega_2, \ldots, \omega_k\}$	a set of k hypothesized clusters, with $\omega_i \subset D$
$C = \{c_1, c_2, \ldots, c_q\}$	a set of q reference clusters, called classes, with $c_j \subset D$
$\lvert A \rvert$	the number of elements in a set A
$N = \lvert D \rvert$	the total number of items in the data set D
$n_{ij} = \lvert \omega_i \cap c_j \rvert$	the total number of items in cluster ω_i which belong to class c_j
μ_i	the centroid of cluster i (cf. equation 5.1)

Jain (2010, p.651) defines automatic data clustering as an instance of *unsupervised [machine] learning* (cf. section 3.1). Using terminology from set theory, the desired solution of a clustering method can be defined as a non-arbitrary *partition* of a set of data items[4], i. e. a split of a data set D into k mutually disjoint, non-empty subsets, with $1 \leq k \leq N$. "Non-arbitrary" in this case means that out of all possible partitions of a set[5], the solution should have the following properties: The partition is required to satisfy a criterion of minimised inter-cluster similarity and maximised intra-cluster similarity of the data items. The size of the clusters is also often required to be balanced in order to avoid trivial or degenerate solutions. Additionally, the number of clusters k is often a parameter of the clustering algorithm.

The data which should be clustered is regarded as a set D of individual data items. A cluster is a block of the partition Ω of D, i.e. it is a subset of the data set which is disjunct from all other blocks in Ω. There is no formal distinction between Ω and C — both are

[4] There are other definitions of data clustering, which do not require the solution to be a partition of the entire data set. For example, it might be allowed for some data items to be members of no cluster, thus, defining a clustering solution as a set of sub-sets which does not need to be exhaustive on the underlying data set (cf. Manning et al., 2009, p.355).

[5] The number of partitions of a set with n elements into k non-empty subsets is the *Stirling number of the second kind* $\left\{ {n \atop k} \right\}$ (cf. Meilă, 2007, p.874; Manning et al., 2009, p.356). It can be defined recursively as $\left\{ {n+1 \atop k} \right\} = k\left\{ {n \atop k} \right\} + \left\{ {n \atop k-1} \right\}$, for $1 \leq k \leq n$ with the initial condition $\left\{ {n \atop n} \right\} = \left\{ {n \atop 1} \right\} = 1$.

specific partitions of a data set D. The solution C is given a priori and it serves as a reference against which the quality of the generated solutions is evaluated by comparing Ω and C. This reference is usually based on a manual annotation of the data set. Reference clusters are commonly referred to as *classes* and hypothesised clusters which are generated by some clustering method are referred to as *clusters* (cf. Rosenberg and Hirschberg, 2007, p.412; Manning et al., 2009, p.356f). This terminological distinction between classes and clusters is also made in this thesis.

Representation and similarity

Data clustering depends on two closely related decisions of the experimenter: the formal representation of the data and a similarity measure. Real data, for example the formant measurements of German vowels from the example shown in figure 5.1, requires some kind of formal representation in order to be processed by a clustering algorithm. The data items are commonly represented in a multi-dimensional feature space where — ideally — similar items naturally tend to cluster together in that space. Usually, a vector space model is used where specific *features* correspond to dimensions of the vector space and the individual data items (e.g. individual measurements) correspond to points within this multi-dimensional feature space. As defined above, the goal of a data clustering method is to group the items of a data set according to their mutual similarity into several disjoint clusters. This requires a definition of *similarity* between the items. If the representation space is defined appropriately, the similarity measure corresponds to a *distance metric* on that space. Similar items are those with a smaller distance between their corresponding points in the feature space. However, similarities between speech item classes derived from perceptual confusion data assume a distance of zero between tokens of the same class, as Mermelstein (1976, p.94) points out[6]. He adds that such similarities can, therefore, only serve as a "guide to the construction of an appropriate distance metric". This implies a possible within-category differentiation of individual tokens.

Clustering and evaluation methods often refer to the *centroid* of a cluster. It is defined for a set of vectors S_i as:

$$\mu_i = \frac{\sum_{d \in S_i} d}{|S_i|} \tag{5.1}$$

5.2.3. Clustering methods

Giving a detailed overview of data clustering methods and their applications is far beyond the scope of this thesis. Instead, I will focus on the basic ideas and present the

[6] As each cell c_{ij} in a confusion matrix contains the number of times a particular stimulus i has been identified as belonging to class j, all responses of the same class are thus integrated and not further distinguished (cf. section 3.6.2).

most relevant concepts with respect to applications in computer experiments in the field of phonetics and phonology.

Two basic types of clustering methods can be distinguished: *hierarchical*, on the one hand, and *partitional* or *flat*, on the other (cf. Steinbach et al., 2000a, p.3; Manning et al., 2009, Chap. 16–17; or Jain, 2010, p.653). Hierarchical clustering methods can be further categorised as *agglomerative clustering* — bottom-up methods which repeatedly merge the most similar clusters — and *divisive clustering* — top-down methods that repeatedly split existing clusters. Hierarchical clusterings can be visualised as trees.

A brief introduction to hierarchical clustering in psycholinguistics is given by Levelt (1970). He points out that hierarchical cluster analysis can be particularly useful in phonology where "the idea of a hierarchical ordering of features is about as old as the concept of feature itself" (Levelt, 1970, p.108). Hierarchical clustering can be carried out such that a complete hierarchy is generated with one all encompassing cluster at the top and all single element clusters at the bottom. Alternatively, a number of desired clusters k can be specified and the clustering algorithm either stops when this number of clusters is reached or it cuts the generated tree such that k clusters are obtained.

Partitional clustering methods, on the other hand, create a single partition of the data set without an associated hierarchical structure. They usually need several passes over the data set in which they repeatedly re-assign items to clusters until a solution is found (or, in general, until a stopping criterion is reached).

According to the given formal definition in terms of classic set theory, each item of the data set is an element of exactly one cluster. Such methods are called *crisp* or *hard clustering*. This term is used primarily to distinguish them from *soft* or *fuzzy clustering* methods which are based on a definition of graded membership of items where a data item can be an element in multiple clusters. Alternatively, they are based on probabilities of items to be a member of a cluster (cf. Manning et al., 2009, p.350,355; or Jain, 2010, p.654).

K-means clustering

One of the oldest, most simple but still most popular clustering methods is the *k-means* algorithm (cf. Jain, 2010, p.653). The algorithm operates on a vector representation of a given data set D, where each data item corresponds to a point $\vec{x}_i$ in a d-dimensional vector space. The centroid is computed for each cluster and the goal of k-means is to minimise the squared error over all k clusters in the set of clusters Ω (cf. Bishop, 2006, p.424f). If μ_j is the mean (the centroid) of cluster ω_j, then the squared error for that cluster is defined as: $J(\omega_j) = \sum_{\vec{x}_i \in \omega_j} ||\vec{x}_i - \mu_j||^2$. The total squared error is then the sum

Algorithm 5.1 Basic k-means algorithm

```
1: Let D = {x⃗_1, ..., x⃗_n} be a set of n points in a d-dimensional metric feature space
2: Let k be the number of desired clusters
3: Select k initial centroids
4: repeat
5:     Assign all points in D to the nearest centroid
6:     Recompute the centroids of each cluster
7: until the centroids do not change
```

of the squared errors over all clusters:

$$J(\Omega) = \sum_{j=1}^{|\Omega|} \sum_{\vec{x}_i \in \omega_j} ||x_i - \mu_j||^2 \quad . \tag{5.2}$$

The k-means algorithm starts by selecting k points as the initial cluster centroids. It then repeatedly assigns all data points to the closest cluster centroid until convergence, i.e. it recomputes the centroids until the clusters stabilise. The basic algorithm is shown in pseudo-code in algorithm listing 5.1. An algorithm based on k-means, the *bisecting k-means* algorithm is presented in section 5.3.2 of this chapter, where it is used in a series of experiments to cluster speech data.

Limitations of K-means

The k-means algorithm requires a number of parameters which determine its behaviour and the generated solutions. The initialisation step of selecting initial centroids is an important aspect for actual implementations. Algorithm 5.1 intentionally does not specify *how* the initial centroids are selected and from which base. Depending on the implementation, this initialisation can be a random selection of points from the data set, or a random selection of arbitrary points in the feature space, etc. Different centroid initialisations can lead to different clustering solutions. One way to counter this effect is to repeat the clustering procedure several times with different, randomly initialised centroid initialisation.

While the parameter k directly determines a global property of the clustering result (the number of clusters), there is no general solution for the optimal selection of k. If k is equal to the number of data points, the total sum of squared error is trivially 0. This means that the mathematically optimal solution is always found if each item is assigned to its own cluster. Which, of course, is not a desired analysis of almost any data set of practical interest. As Jain (2010, p.652) points out, "in reality, a cluster is a subjective entity that is in the eye of the beholder and whose significance and interpretation requires domain knowledge." Therefore, the selection of k requires domain knowledge

and it might be necessary to test different values for k by comparing and interpreting the resulting data partitions according to different parameter settings. This problem is also visualised in figure 5.1 (discussed above). The parameter was set to $k = 15$ based on the knowledge that the data represent formant measurements of German vowel phonemes and the expectation that there should be 15 clusters in the data set. Ignoring this a priori knowledge and looking only at the raw data distribution (left panel in figure 5.1), other numbers of clusters could be equally plausible. The necessary a priori definition of the number of clusters counters the purpose of cluster analysis as a tool for exploratory investigations of unknown structures in a data set. Clustering experiments can be employed to investigate and compare alternative hypotheses about the structure of a data set and to question existing expectations[7].

The example in figure 5.1 on page 162 illustrates an important feature of k-means: it always generates contiguous (hyperspherical) clusters while the true clusters might show considerable overlap as in that case. In general, the selection of a particular similarity measure (usually a distance metric in the d-dimensional feature space) determines the shape of the found clusters.

Fuzzy clustering

The definitions in section 5.2.2 refer to hard clustering methods. An alternative to creating a partition on a data set is to employ principles based on fuzzy set theory, where the membership of an item is not a binary but a continuous-valued function.

Erriquez et al. (2000) use a fuzzy clustering neural network approach for acoustic-phonetic modelling in ASR. This was motivated by the observation that the categories in speech data is not uniformly distributed and that it shows considerable overlap (p.35f). A clustering experiment based on MFCCs is reported to have failed. The authors speculate that this failure might be due to the non-hyperspherical nature of acoustic clusters and due to "the fact that in speech, phones are not realised as clearly distinct clusters" (p.37). They compare their implementation to a SOM and find that the SOM is superior with respect to classification accuracy (cf. section 3.4.1 on page 70).

The vowel space, for example, is characterized by considerable overlap of the vowel categories. Whether this overlap suggests an underlying *fuzzy* representation is not clear. Goldsmith and Xanthos (2009, p.26) speculate about the applicability of fuzzy clustering for the classification of phonemes which "in effect belong to more than one group". See for example Aarts's (2004) historical overview of the concepts of *gradience* and *vagueness* in linguistics, and, in particular, the interpretation of vagueness that the exact location of a boundary between two categories might just happen to be unknown or varying, but still discrete (e.g. p.347). In exemplar-theoretic approaches, the exemplar clouds

[7] "Wie gesagt: denk nicht, sondern schau!" (*As said: don't think, but look!*, translation: DD; Wittgenstein, 1971, §66/p.57).

representing linguistic categories might overlap while the category associated with each individual exemplar is still discrete (cf. section 6.1 on page 217 for more details on exemplar theory). Thus, it is an open question whether it is more appropriate to apply fuzzy clustering methods to speech items or whether it is sufficient to use hard clustering methods which do not necessarily create hyperspherical clusters.

5.2.4. Evaluation measures for data clustering

What constitutes the main motivation for the application of a clustering method is also a severe problem when it comes to evaluation of clustering methods and the assessment of the quality of an obtained solution: the true distribution and/or categorisation of the data items is unknown. Levelt (1970, p.107) discusses an "overlap measure" for the evaluation of hierarchical clusterings and points out the problem that the real distribution of the data is unknown and that "significance tests cannot be applied." He refers to a study using Monte Carlo simulation for evaluation and concludes: "Without computer simulations one must rely on an intuitive evaluation [...]"

Evaluation of computer simulation experiments in general is discussed in section 3.6 on page 74. This section focuses on the evaluation of data clustering solutions and commonly employed evaluation measures in this context. As other methods discussed in this thesis, solutions of automatic data clustering methods can be evaluated either *directly* or *indirectly*. Indirect evaluations assess the performance of the overall system which makes use of a clustering method as one of its data processing modules. Examples of studies where such an evaluation of a clustering solution is done are the experiments on speech segmentation by Pereiro Estevan et al. (2007) and Scharenborg et al. (2007)[8]. Such indirect evaluation methods, however, will not be further discussed in this section.

Direct evaluations of clustering solutions can be further classified into two different types of evaluation measures: these are *external* and *internal* evaluations[9] (for discussions of cluster evaluation see for example: Steinbach et al., 2000a; Meilă, 2007; Rosenberg and Hirschberg, 2007). External, measures compare a clustering solution against a reference partition. An internal evaluation, on the other hand, does not refer to external knowledge to assess the quality of a clustering solution.

There is no common agreement on what criterion is most useful to assess the quality or validity of clusterings. It has been pointed out that one must not expect to find

[8] See sections 4.1 and 4.2 for more details.

[9] Jain (2010, p.657) mentions three different types of evaluation measures for data clustering (or "cluster validity indices"): *internal*, *relative*, and *external*. Relative measures evaluate several clustering solutions against each other and determine which one is the best. This decision, however, can only be based on data intrinsic criteria or on a comparison against some external reference criteria. Therefore, I will assume such measures could be classified as either *internal* or *external* measures, and I will not discuss them separately.

a general purpose evaluation measure for any kind of clustering task, e.g. by Meilă (2007):

> "Just as one cannot define a 'best' clustering method out of context, one cannot define a criterion for comparing clusterings that fits every problem optimally."
> (Meilă, 2007, p.891)

As a complete overview of cluster analysis in general cannot be given here, the review of evaluation measures in this section is also limited to the most relevant evaluation measures which have been applied in computer experiments on clustering speech data.

Internal evaluation measures

In cases where no reference data is available, the quality of a clustering must be assessed based on the intrinsic properties of the partition Ω, for example by evaluating the item-to-item similarities within the clusters and/or between clusters. A possibility is determining the cohesiveness of the clusters. This can be computed using the weighted internal cluster similarity, as defined by Steinbach et al. (2000a, p.8):

$$\frac{1}{N^2} \sum_{d \in c_i, d' \in c_i} \cos(d', d) \quad , \tag{5.3}$$

which is equivalent to the squared length of c_i's centroid $||\mu_i||^2$.

Internal evaluation measures can directly employ the objective function which a clustering algorithm attempts to optimise: the criterion of minimised inter-cluster similarity and maximised intra-cluster similarity of the data items. A detailed study on the comparison of different internal evaluation measures for data clustering is provided by Gurrutxaga et al. (2011). They examine previous studies which compare different evaluation measures (also called "cluster validity indices") by comparing their respective scores for generated partitions for various settings of k, the main criterion being the successful detection of the correct number of clusters. The authors argue that these earlier studies on evaluation measures have been based on the assumption that a clustering algorithm produces the best partition if the number of clusters is equal to the true number of classes — and that this assumption is false in general (p.506). They argue for a standardised framework for the comparison of evaluation measures and propose a methodology which does not focus on the evaluation measures' ability to detect the correct number of clusters but on their ability to identify the partition most similar to the reference solution[10]. Their results show that the compared internal evaluation measures are much better in detecting the partition most similar to the reference than in detecting

[10]Evaluation and comparison of evaluation measures in general is discussed in sections 3.6 (page 74). Section 4.3 (page 110) presents a meta-evaluation study for the case of speech segmentation.

the correct number of clusters (p.512). The data sets on which Gurrutxaga et al. (2011, p.510f) based their study have between 2 and 8 classes and a maximum of 336 data points. Most of the data sets were synthetic and no speech data was among the "real" data sets. It thus remains to be investigated, whether their conclusions can be carried over to the domain of phonetics and phonology.

The clustering experiments presented in sections 5.3 and 5.4 in this chapter evaluate the clustering solutions against a given reference. Therefore, internal measures will not be further discussed here.

External evaluation measures

An external evaluation refers to a given reference assignment of each item of the data set to a specific reference class (e.g. the phonetic labels assigned to a set of speech segments). Using the terminology defined above, an external evaluation compares a clustering solution Ω against a given reference C. While it is a trivial task to determine whether a clustering solution is perfect according to an external reference, Rosenberg and Hirschberg (2007, p.410) point out that it is rather difficult to determine "how far from perfect an incorrect clustering solution is". Since they are based on the comparison of two partitions, external evaluation measures can also be called *partition similarity measures* (Gurrutxaga et al., 2011, p.508). The following parts of this section present some often used external evaluation measures.

Contingency tables and classification accuracy

Most external evaluation measures for data clustering are based on the construction of a contingency table or confusion matrix (Meilă, 2007, p.875), cf. section 3.6.2 on page 76. In the case of data clustering, the contingency table tabulates the number of data items for each cluster in Ω against their corresponding reference classes in C. The cell (i, j) of the contingency table contains the number of items n_{ij} in cluster i that belong to class j, i.e. $n_{ij} = |\omega_i \cap c_j|$, as defined above.

Without loss of generality, I adopt the (completely arbitrary) convention that columns in the contingency table correspond to classes and rows correspond to clusters. The order of the rows (clusters) is not a priori related to the order of the columns. If one arranges the rows such that the cells with the maximum value of each cluster lie on the matrix diagonal (or close to it), one can immediately associate the clusters with the classes, and the contingency table may be interpreted as a confusion matrix. If less clusters are produced than there are classes, one can obtain a square confusion matrix by adding an empty row to the contingency table and interpret one class as not reproduced by the clustering method. I propose two different methods of constructing a confusion matrix for the evaluation of a clustering solution in section 5.3.

The most simple measure of clustering quality is *classification accuracy* which is directly based on the corresponding confusion matrix. In a classification task, the classification accuracy can be defined as the sum of correctly classified items divided by the total number of items; formally:

$$\text{classification accuracy} = \frac{1}{N}\sum_{i=1}^{k} n_{ii} \quad , \tag{5.4}$$

where n_{ii} is the number of items in hypothesised category i which actually belong to reference category i, and k is the number of classes, both, hypothesised and true. In terms of the confusion matrix, the classification accuracy corresponds to the trace of the matrix (i.e. the sum of its diagonal entries), divided by the total sum of entries.

According to the terminology defined in section 3.6.2 on page 76, the results of a clustering solution Ω can always be represented in a contingency table relative to a given reference C. In the general case, however, this contingency table cannot be interpreted as a confusion matrix. Clustering is not a classification task. The number of clusters does not need to be equal to the number of classes. Also, the association between clusters is usually ambiguous. It depends on the definition of an arbitrary mapping from clusters to classes which is not necessarily a one-to-one mapping. Data clustering groups items according to their mutual similarities, it does not assign labels to the clusters. Therefore, a classification accuracy can only be reported if clusters are mapped to classes, which is equivalent to the construction of a confusion matrix.

Partition Entropy

Entropy is an information-theoretic measure for the amount of information contained in a given message (cf. section 3.6.4). Entropy can be applied to data clustering as an external clustering evaluation measure computing the entropy H_i of each cluster ω_i according to a given reference. It can be defined as follows: Given two partitions, a clustering solution Ω and a reference C, the entropy associated with each cluster ω_i is $H(\omega_i) = -\sum_j p_{ij} \log p_{ij}$, where p_{ij} is the probability of a randomly chosen item $d \in D$ to be a member of class c_j and cluster ω_i. The total entropy $H(\Omega)$ of a clustering solution Ω can then be defined as:

$$H(\Omega) = \sum_{i=1}^{k} \frac{|\omega_i| \cdot H(\omega_i)}{N} \quad , \tag{5.5}$$

where $|\omega_i|$ is the number of items in cluster i, $k = |\Omega|$ the number of clusters and N the total number of items (Steinbach et al., 2000a, p.6f)[11].

[11] Original notation changed for consistency.

Partition F-measure

F-measure is a widely used external evaluation measure for a variety of tasks in information retrieval, machine learning or NLP, as discussed in section 3.6.3 on page 77. In the domain of data clustering, the basic quantities precision and recall can be defined for a cluster ω_i and a class c_j as follows (adopting the definitions from Steinbach et al., 2000a, p.7 and Rosenberg and Hirschberg, 2007, p.412f; my notation):

$$\text{precision}(\omega_i, c_j) = \frac{|\omega_i \cap c_j|}{|\omega_i|} \tag{5.6}$$

and

$$\text{recall}(\omega_i, c_j) = \frac{|\omega_i \cap c_j|}{|c_j|} \quad , \tag{5.7}$$

where $|\omega_i \cap c_j|$ is the number of items in cluster ω_i which belong to class c_j according to the terminology defined in section 5.2.2.

If the F-measure for a cluster ω_i and a class c_j is defined as usual as

$$\text{F}(\omega_i, c_j) = \frac{2 \cdot \text{precision}(\omega_i, c_j) \cdot \text{recall}(\omega_i, c_j)}{\text{precision}(\omega_i, c_j) + \text{recall}(\omega_i, c_j)} \quad , \tag{5.8}$$

then the overall F-measure for a clustering solution Ω can be defined as follows:

$$\text{F}(\Omega, C) = \sum_j \frac{|c_j|}{N} \max_i \{\text{F}(\omega_i, c_i)\} \quad . \tag{5.9}$$

Thus, the F-measure for a clustering solution is defined as the weighted sum of maximum F-measure for each class over all clusters. Note that Steinbach et al., 2000a, p.7 define the partition F-measure for hierarchical clusterings, although it can be applied to flat clusterings as well.

Adjusted Rand Index

The basic idea behind the Rand index is that two partitions can be compared by pair-wise comparison of their N items (Hubert and Arabie, 1985, p.194f). Given two partitions $\Omega = \{\omega_1, \ldots, \omega_K\}$ and $C = \{c_1, \ldots, c_J\}$ of a set D, there are four possible cases which can be distinguished for any pair of items (d_i, d_j), with $d_i \in D$, $d_j \in D$:

- $d_i \in \omega_g \wedge d_j \in \omega_g \wedge d_i \in c_h \wedge d_j \in c_h$, i.e. both items are elements of the same class and of the same cluster — let the total number of such pairs be TP.
- $d_i \in \omega_g \wedge d_j \notin \omega_g \wedge d_i \in c_h \wedge d_j \notin c_h$, i.e. both items are elements of different classes and clusters — let the total number of such pairs be TN.

- $d_i \in \omega_g \wedge d_j \notin \omega_g \wedge d_i \in c_h \wedge d_j \in c_h$, i.e. both items are elements of the same class but belong to different clusters — let the total number of such pairs be FN.
- $d_i \in \omega_g \wedge d_j \in \omega_g \wedge d_i \in c_h \wedge d_j \notin c_h$, i.e. both items are elements of the same cluster but belong to different classes — let the total number of such pairs be FP.

TP and TN are cases where two partitions agree, and FN and FP are cases where two partitions disagree[12]. In general, if two partitions are similar, they will be associated with larger values of agreements and smaller values of disagreement. Hubert and Arabie compare different cluster quality measures based on raw counts of agreements and disagreements and they point out that such "raw indices are difficult to evaluate and compare since they are neither measures of departure from a common baseline nor are they normalised to lie within certain fixed bounds (e.g., 0 and 1 or ±1)" (p.197).

Based on these definitions of four types of item pairs and their total counts, and based on the general form of an index corrected for chance as

$$\frac{\text{index} - \text{expected index}}{\text{maximum index} - \text{expected index}} \quad ,$$

Hubert and Arabie (1985, p.198) propose a Rand index corrected for chance, which is defined for two partitions Ω and C as follows[13]:

$$\mathrm{ARI}(\Omega, C) = \frac{\sum_{i,j} \binom{n_{ij}}{2} - \sum_i \binom{|c_i|}{2} \sum_j \binom{|\omega_j|}{2} / \binom{N}{2}}{\frac{1}{2}[\sum_i \binom{|c_i|}{2} + \sum_j \binom{|\omega_j|}{2}] - \sum_i \binom{|c_i|}{2} \sum_j \binom{|\omega_j|}{2} / \binom{N}{2}} \tag{5.10}$$

Alternatively, the adjusted Rand index (ARI) can be defined based on the four sets of pair types as follows (Duran et al., 2010b, p.148):

$$\mathrm{ARI}(\Omega, C) = \frac{2(\mathrm{TP} \cdot \mathrm{TN} - \mathrm{FP} \cdot \mathrm{FN})}{(\mathrm{TP} + \mathrm{FP})(\mathrm{FP} + \mathrm{TN}) + (\mathrm{TP} + \mathrm{FN})(\mathrm{FN} + \mathrm{TN})} \tag{5.11}$$

The adjusted Rand index is corrected for chance such that the value has an upper bound of 1 for a perfect solution and a value of 0 for the expected chance level solution. Negative values are interpreted as being of "no substantive use" (Hubert and Arabie, 1985, p.198).

Purity

Purity is an external evaluation measure based on matching clusters to reference classes. Each cluster is assigned to the class for which the intersection with the cluster contains

[12] See also section 3.6.1 on page 76 for basic terminology.
[13] Notation adapted according to the definitions in section 5.2.2 on page 161.

the most items — I will refer to this as the "majority class" of a given cluster. Purity is equal to the sum of items associated with a majority class over all clusters divided by the total number of items N in the data set D (cf. Strehl, 2002, p.107f; Rosenberg and Hirschberg, 2007, p.412; Manning et al., 2009, p.356f)[14]. For a given set of clusters Ω and a set of classes C, purity can be formally defined as follows:

$$\text{purity}(\Omega, C) = \frac{1}{N} \sum_{i=1}^{|\Omega|} \max_j (|\omega_i \cap c_j|) = \frac{1}{N} \sum_{i=1}^{|\Omega|} \max_j n_{ij} \tag{5.12}$$

Purity has an upper bound of 1 for a perfect solution and a positive value close to 0 for bad solutions. The lower bound of purity is $\frac{1}{N}$, which is achieved only for the degenerate case where each item in D belongs to its own class, and all items are grouped into a single cluster. A trivially perfect score is achieved by a clustering solution which assigns each item to its own cluster. The total number of clusters is not compared to the number of reference classes. Due to this bias towards larger numbers of clusters, purity cannot be used to compare clustering solutions with different numbers of clusters.

V-measure

The *V-measure* is proposed by Rosenberg and Hirschberg (2007)[15] as an entropy-based external clustering evaluation measure which does not depend on the number of clusters. The authors emphasise that it is specifically designed to address the problem of quantifying "how far from perfect" a clustering solution is (p.410).

V-measure is defined similar to the F-measure as the harmonic mean of *homogeneity* and *completeness* (Rosenberg and Hirschberg, 2007, p.411f) as follows:

$$\mathrm{V}_\beta(\Omega, C) = \frac{(1+\beta) \cdot h \cdot c}{(\beta \cdot h) + c} \tag{5.13}$$

[14] An alternative definition of "*cluster purity*" which is sometimes used in the literature on speech processing is given by Solomonoff et al. (1998, p.758) as follows (notation adapted for consistency):

$$\textit{cluster purity}(\omega_i) = \sum_{j=1}^{|C|} \frac{|\omega_i \cap c_j|^2}{|\omega_i|^2}$$

The average for all clusters can then be used as a measure for the overall quality of a clustering solution Ω. This definition of "cluster purity" is not further considered in this thesis, and the term "purity" will exclusively refer to the definition given in equation 5.12.

[15] A Java implementation by Andrew Rosenberg can be found online at `http://eniac.cs.qc.cuny.edu/andrew/`. This implementation is not used in the experiments in sections 5.3 and 5.4. However, the values returned by my own implementations of *purity* and *V-measure* (cf. section B.2.1) are identical to those returned by the methods implemented in this Java library.

where homogeneity h is defined as

$$h = \begin{cases} 1 & \text{if} \quad H(C,\Omega) = 0 \\ 1 - \frac{H(C|\Omega)}{H(C)} & \text{else} \end{cases} \tag{5.14}$$

and completeness c is defined as

$$c = \begin{cases} 1 & \text{if} \quad H(\Omega,C) = 0 \\ 1 - \frac{H(\Omega|C)}{H(\Omega)} & \text{else} \end{cases} \tag{5.15}$$

where $H(X)$ is the entropy of a set X (cf. Manning et al., 2009, p.358), $H(X,Y)$ is the joint entropy, and $H(Y|X)$ is the conditional entropy, with:

$$H(C|\Omega) = \sum_{i=1}^{|\Omega|} \sum_{j=1}^{|C|} \frac{n_{ij}}{N} \log \frac{n_{ij}}{|\omega_i|} \tag{5.16}$$

$$H(C) = \sum_{j=1}^{|C|} \frac{|c_j|}{N} \log \frac{|c_j|}{N} \tag{5.17}$$

$$H(\Omega|C) = \sum_{j=1}^{|C|} \sum_{i=1}^{|\Omega|} \frac{n_{ij}}{N} \log \frac{n_{ij}}{|c_j|} \tag{5.18}$$

$$H(\Omega) = \sum_{i=1}^{|\Omega|} \frac{|\omega_i|}{N} \log \frac{|\omega_i|}{N} \tag{5.19}$$

As with the F-measure, the parameter β in equation 5.13 allows assignment of different weights on homogeneity or completeness. I will set $\beta = 1$ in the remainder of this thesis. The definition of the V-measure ensures that good solutions get values closer to 1 and bad solution values closer to 0. Compared to purity, the V-measure has some advantages: It is not based on a simple mapping of each cluster to a reference class, and it is independent of the actual number of classes and the size of the data set.

5.2.5. Examples of cluster analysis in Phonetics and Phonology

Data clustering is frequently used in NLP tasks. It has also been applied in speech processing. In the domain of ASR, for example, Solomonoff et al. (1998, p.758), apply an agglomerative clustering method to group speech recordings by speakers. Similarly, Qiao et al. (2008, p.3989f) optimise an objective function for automatic

speech segmentation by applying an agglomerative frame-by-frame clustering. An early example of data clustering applied to speech data is the study by Dammann (1966). The data is taken from recordings of eight English vowels, 250 data points in total. It is, however, primarily meant as a "practical test" for an earlier published clustering method (p.80).

Goldsmith and Xanthos (2009, p.11ff) use a "spectral clustering" method to learn phonological categories. This is based on a symmetric similarity matrix for all items in the data set on which operations of linear algebra are applied to create a partition. They apply the clustering method to symbolic data (phonemes) and present experiments to learn the distinction between the categories vowel and consonant (p.15f), or vowel harmony (p.24).

A clustering methods is often used as one processing step besides others in larger simulation experiments with continuous data. One such study is for example presented by Scharenborg et al. (2010). They present a method of unsupervised speech segmentation which is based on *maximum margin clustering*. The principle behind this clustering method is "maximum margin classification", i.e. the maximisation of the margin between the data points of the clusters (Xu et al., 2005, p.1538f). The margin is the distance of the nearest data point(s) of a cluster to a hyperplane which separates the clusters in a usually high dimensional representation space (Yoshida and Sakurai, 2002, p.111). They do not evaluate the clustering results directly but evaluate the resulting speech segmentation against a reference segmentation[16]. An exemplar-theoretic model of speech production is presented by Kirchner and Moore (2009). A hierarchical clustering step is used to test the similarity of the model's exemplar clouds[17].

Rosenberg and Hirschberg (2007) present a clustering experiment on pitch accent types. This experiment is primarily presented as a demonstration of the *V-measure* proposed by the authors. A more detailed investigation on pitch accent types is presented by Lintfert et al. (2011). They use k-means clustering to investigate the acquisition of prosodic categories. The clustering is applied primarily to group similar realisations such that the cluster centroids can be interpreted as "prototypical" realisations (p.758). The parameter k is systematically varied. Lintfert et al. point out that the advantage of such a clustering approach is that it allows the investigation of intermediate categories at different developmental stages. This provides an additional level of analysis which is not restricted to the direct comparison of the individual productions against the adult category system (p.759). Schweitzer (2010b, p.118ff) presents simulation experiments on human categorisation of prosodic events. Detection of prosodic categories is modelled by a clustering procedure. Different clustering methods are applied to prosodic data extracted from a speech database (p.124).

[16]See chapter 4 on page 83 for a detailed discussion on speech segmentation methods and experiments and their evaluation.

[17]See section 6.1 for more information on exemplar theory.

Clustering methods are often applied in simulation studies on speech and language acquisition. Lin (2004, 2005) describes a clustering-based approach for learning phones directly from the speech signal in which phonetic features are automatically learned. Coen (2005, 2006) presents a "cross-modal" clustering approach and applies it to the task of vowel category acquisition (cf. section 2.2.1). Howard and Messum (2011) apply a clustering method to group motor patterns. Similar to the Lintfert et al. (2011) study, they keep the most central exemplar. The model implements a perception–production loop and operates on acoustic and articulatory data. A variant of the k-means method is applied to acoustic data which can handle patterns of different lengths (Howard and Messum, 2011, p.104).

5.2.6. Discussion and conclusions

Data clustering can be a powerful tool for unsupervised organisation of data. It has been used by many authors in simulation studies in the field of phonetics and phonology addressing a wide range of research questions.

Cluster analysis can be a useful tool in computer simulation experiments in phonetics and phonology if, for example, no set of given categories should be assumed. It can also be applied to investigate whether a given set of linguistic categories can be derived from some low-level similarity relations on the underlying data and what the nature of these similarity relations might be. By using automatic clustering in a phonetic simulation experiment, the situation can be modelled in which the learner has to make its own decisions about the category of a new, unknown item.

Problems

Coen (2006, p.1453) points out that the task of clustering "perceptual data is highly non-parametric in that *both the number of clusters and their underlying distributions are unknown*."[18] Clustering solutions are biased by the specific properties of a given clustering method, especially by its parameters. The selection of parameter settings is in turn biased by expectations of the researcher.

The number of clusters. The mathematically optimal solution is often the one which assigns all data items to one cluster. Thus, maximum margin clustering, for example, requires a constraint to balance the sizes of the generated clusters since the margin is trivially maximal if all data items are assigned to one cluster (Xu et al., 2005, p.1539). This will be rarely, if ever, considered an acceptable solution. The number of clusters is a critical parameter for many clustering methods, e.g. the widely used k-means clustering. This determines greatly the global structure of the solution. The number of clusters can

[18]Emphasis in the original.

be explored by testing a range of different values and comparing the respective solutions, or it can be set to an expected value based on linguistic knowledge (e.g. the number of categories expected to be found in a data set).

Data representation. Clustering methods require specific representations of the analysed data. Often, it is represented in a high-dimensional vector space where similarity can be defined easily by means of a distance metric. Apart from increasing computational complexity with increasing dimensionality, such representations suffer from the problem that they a not accessible to human intuition. Also, the shape of the found clusters depends on the representation space and the similarity measure. If it is known in advance that the true clusters in the data are non-convex, an appropriate clustering method can be selected which is able to handle such structures — which classic k-means cannot.

Similarity. Another a priori decision that needs to be made by the experimenter is the definition of an appropriate similarity measure. Many cluster analysis methods depend on a metric defined on the data set. Representation of speech data and similarity measures depend on each other. For symbolic data (i.e. data with discrete features) as well as for continuous data, defining an appropriate metric is a non-trivial problem.

Evaluation and reference data. Evaluating the results of the applied clustering method, Dammann points out the inherently subjective nature of manually created reference classes in data taken from speech recordings:

> "In a case such as the present experiment where the patterns emerge from the real world rather than from a pattern-generating rule, there can be no assurance that any particular sample belongs to its parent class in accordance with any reasonable objective criterion. [...] In determining unknown classes the experimenter is faced with the task of sequentially separating the sample space into groups of patterns. The problems of where to make these separations and how long to continue to make separations both arise."
> (Dammann, 1966, p.87f)

From an engineering point of view, there is no solution to this last mentioned problem. In phonetics and phonology, the decision of where to draw the line depends on the particular linguistic level of linguistic abstraction (e.g. the distinction between broad and narrow transcriptions of speech). The problem of the inherent subjectiveness in given reference data is also discussed in chapter 4.

If clustering is seen as an exploratory data analysis technique, then using it, for example, for grouping of speech segments, presupposes that there is no assumption made about the actual structure of the data, e.g. the inventory of segment types. In confirmatory studies, clustering can be used to test certain models on a corresponding data set. It is, however, difficult to do confirmatory studies with automatic data clustering

since the definition of the underlying similarity measure might be (partially) unknown. Exploratory studies, on the other hand, require some appropriate parameter settings and it might be necessary to test a variety of parameters and evaluate the results for each setup.

Data clustering seems to be particularly suitable for models based on Exemplar Theory (cf. section 6.1). A central concept in exemplar-theoretic accounts of human speech production and perception is the existence of a similarity measure which groups given exemplars according to their mutual degrees of similarity and thus generates various clusters or exemplar clouds. Clustering can be used in this context as a means to simulate (or approximate) the human ability of recognising similarity between distinct items and grouping these items according to their perceived similarity. This comparison is, however, partially inadequate. An automatic clustering procedure partitions a data set into non-overlapping subsets which completely cover the entire data set. In exemplar theory, "clouds" of exemplars which (implicitly) form classes or categories might overlap, and not all exemplars need to be associated with a particular category. Within the framework of exemplar theory it might seem more appropriate to consider a fuzzy clustering method instead of a hard clustering.

Manually annotating a data set, e.g. a speech signal, implicitly defines a clustering solution to the data such that each data point is assigned to a particular class. In many NLP tasks, data is available in such abundance that a complete manual annotation is too expensive. Applying unsupervised methods such as data clustering to a data set can reduce the amount of required manual work, but it also adds the need to evaluate the quality of clustering solution. When defining a model of speech perception and recognition, for example, the use of some pre-defined data annotations is often not wanted. However, a manually created annotation is usually required to evaluate the clustering results. A variety of evaluation measures was discussed in section 5.2.4.

5.3. Experiment 1: Clustering phonetic segment data

In this section, I present a simulation experiment on clustering acoustic and articulatory data. The basic experiment and its first results have been presented previously by Duran et al. (2011a) at the INTERSPEECH conference.

Humans face the task in first language acquisition of recognising patterns within the continuous, highly ambiguous and variable speech stream, and to partition it into linguistically relevant units. Segmentation of the speech stream and identification of individual speech units is one of the first steps in the acquisition of linguistic categories. However, at the acoustic level there are no two identical utterances. Some method is therefore required to group individual speech events according to their mutual similarities into distinct groups or categories. The term *speech event* is used here to refer to any linguistic units such as phones, phonemes, syllables, words, phrase or entire utterances.

The segmentation problem has been discussed in detail in chapter 4. There, the notion of *boundariness* has been used to describe the fuzziness inherent in linguistic boundaries as well as the graded nature of linguistic units. This chapter is concerned with clustering of speech events. The terminus technicus *clustering* seems more appropriate for the issues discussed here then the commonly used terms *recognition* or *categorisation*. One of the fundamental features of first language acquisition is that it is essentially an unsupervised bottom-up process, as is argued in this thesis. The speech items have to be analysed and grouped based only on data internal, implicit or indirect information. Meta-information like segment boundary locations or category labels are not available to infants (cf. section 3.1 on page 65).

The formed clusters of similar speech items could form a first step towards categorisation of short stretches of speech. A serious problem for such bottom-up analyses of the acoustic speech signal is the non-stationarity observable in speech segments. There may be, for example, no purely acoustic cues that would group a sequence of three auditory events like: SILENCE, PLOSION, RELEASE into a single phonetic segment. However, the fact that the entire sequence is caused by one dynamic articulatory gesture will bias any learning algorithm to view them as a single phone. Articulatory information is incorporated in some computational models of speech perception/production, as discussed in section 5.1 of this chapter.

The experiments presented in this section address the question of segmental category information provided by the acoustic signal on the one hand and the recorded articulatory movements on the other. Additionally, the experiments address the questions whether the full continuous EMA signals can be used analogously to the usage of acoustic data in computational models of speech perception, without the need of manual annotation of articulatory gestures or landmarks.

The remainder of this section is structured as follows: The basic clustering experiment is presented first, followed by a comparison of the results against a baseline system presented in section 5.3.4. Finally, a refinement of the basic experiment is presented in section 5.3.5. Algorithm 5.2 gives a compact overview of the experiment in pseudo-code.

5.3.1. Speech material: Polish EMA corpus

The speech data for this study has been taken from a Polish EMA corpus. This corpus contains acoustic and articulographic recordings taken from three native speakers (one male, two female). Details about this database are given in appendix A.1 on page 251. The EMA measurements are sampled at 250 Hz — which corresponds to 1 frame for every 4 ms. Four EMA signals are used: lip distance, tongue tip and two for tongue body movements. These signals are combined such that each frame forms an eight-dimensional vector with each dimension corresponding to one EMA measurement in the horizontal and in the vertical planes. The acoustic data has been converted to

Algorithm 5.2 Clustering experiment

```
Require: Subset D of Polish EMA corpus for one speaker
Require: Sampling function f_sample
Require: Clustering function f_cluster
for all data types ∈ {AC, AR, ACAR} do
    Determine class size s
    for r iterations do
        Define corpus sample D' = f_sample(D) with s items from each class
        Compute and evaluate f_cluster(D')
    end for
    Compute average result over all r iterations
end for
```

provide a structurally similar representation. Amplitude envelopes were taken from eight logarithmically spaced frequency bands. This representation was chosen according to earlier work by Wade et al. (2010, p.232) on the Context Sequence Model (CSM) of speech production[19]. The representation of the acoustic speech signal by envelopes from logarithmically spaced frequency bands was inspired by work by Loizou et al. (1999, p.2098) on the number of channels required to understand speech. They found that "eight channels are needed to reach asymptotic performance" in a perception experiment (p.2102). The choice of using such a representation in the experiment presented here is particularly based on the idea to reduce the amount of signal processing.

5.3.2. Method and implementation

The experiment is implemented and carried out in MATLAB, version 7.8.0 (R2009a), on a 64-bit Linux system. The commented MATLAB source code is listed in the appendix in section B.2.1, starting on page 330.

The experiment presented by Duran et al. (2011a) differs in some details of the implementation from the experiment presented here. It used, for example, a Python implementation of the *bisecting k-means* algorithm (see section 5.3.2) for the actual clustering step. For the experiment presented here, the bisecting k-means clustering procedure has been implemented in MATLAB (cf. source code listing B.42 on page 347).

The corpus was pre-processed by extracting all data from the original EMU files (cf. section A.1), converting it as described in the previous section and storing it in binary format for faster data access and more efficient processing.

The initialisation script `clusteringExperiment1` is shown in source code listing B.37. It sets all experimental parameters, initiates logging of output and starts the main

[19]See also chapter 6 for more details on the CSM and my own experiments based on it.

function of the experiment. The main experimental parameter is the *speaker* since the clustering procedure is applied only to single-speaker data. Another experimental parameter is *data type*. The experiment is repeated for three data type conditions: (1) using articulatory EMA data, (2) using acoustic data and (3) using a combination of both EMA and acoustic data. These data types are denoted by the symbols AR, AC and ACAR in Algorithm 5.2 on page 181, respectively. The speech data of all phonetically labelled parts was extracted from the Polish EMA corpus, comprising a total of 6263 frames of speech material of speaker 1, 7375 frames for speaker 2 and 7427 frames for speaker 3. All data frames are associated with the phone label of the corresponding phonetic segment. Formally, all data frames s_i with a time index $t_h < t_i \leq t_j$ between two segment labels at index t_h and index t_j are labelled with the phonetic label c_i at index t_j, where t_h is the index of the immediately preceding phone label c_{i-1}, or 0 for the beginning of the data sequence[20].

A reference phone class is defined as the set of all data frames which are labelled with the same phone label. The set of class labels used for the experiment corresponds to the set of phone labels $C = \{\mathrm{p}, \mathrm{r}, \mathrm{a}, \mathrm{l}, \mathrm{k}, \mathrm{i}, \mathrm{ɨ}, \mathrm{u}\}$ in the Polish EMA corpus (given here in IPA transcription). The class label /ɨ/ corresponds to orthographic ⟨y⟩, the Polish central high vowel. Since the sets of data frames which correspond to particular labels are not of equal size (cf. table 5.1), a random sampling procedure is applied to generate data sets with equally sized phone classes. This random sampling procedure was repeated ten times for the work presented by Duran et al. (2011a). The results presented here are based on 1000-fold clustering on random samples of the corpus data. The initialisation script defines a handle to the function which generates the random sub-sets of the corpus. In section 5.3.5 this functionality is used to define a different sampling function without the need to re-implement the entire experiment.

The defined parameters are passed to the actual experiment which is implemented in function `experiment1` (see listing B.38 on page 332). The function implements a number of nested loops which iterate over the experimental parameters and repeat the clustering procedure for each parameter combination. The outer-most `for`-loop runs over all single-speaker sub-corpora such that no data is mixed from different speakers. Embedded in the speaker loop is a loop which iterates over the different data type settings. For each of the data types, the clustering procedure is repeated for a set number of times (here: 1000) on a sample from the corpus. Each individual clustering solution is evaluated and the results are accumulated and averaged over all repetitions.

[20]The segment labels in the Polish EMA corpus are assigned according to the EMU scheme to the end index of the corresponding segment.

	class									class
speaker	k	p	r	l	u	ɨ	i	a	total	size
1	1319	1355	674	613	147	278	586	1291	6263	110
2	1327	1477	593	960	154	340	622	1902	7375	116
3	1577	1647	744	719	139	375	655	1571	7427	104

Table 5.1.: Number of frames per phone class

Corpus data sampling

To generate a random sample set, the *minimum cluster ratio* is set to 75% (a technical parameter of the implementation). The number of frames which needs to be taken from each phone class is determined according to the phone class with the smallest total number of frames in the corpus. This corresponds to 110 frames per phone class for speaker 1, 116 frames per class for speaker 2 and to 104 frames per phone class for speaker 3. Table 5.1 shows the number of frames for each phone class in the three speaker sub-corpora and the corresponding class sizes. For speaker 1, for example, the smallest phone class is the set of frames labelled /u/, with a total of 147 frames. Taking 75% of this number (rounding to the nearest integer), the size of the classes for the sample sets is set to 110. Therefore, sample sets for speaker 1 are generated by the function `getSampleSet` such that all eight phone classes are represented by a set of 110 frames taken randomly from the corpus (cf. listing B.40).

Bisecting k-means

The *bisecting k-means* method is used for data clustering in the experiments presented in this thesis (Steinbach et al., 2000b). This is a variant of the frequently used *k-means* algorithm (cf. section 5.2.3). The bisecting k-means algorithm takes an existing cluster and bisects it using the basic k-means method. This is performed for a specified number of iterations on each cluster and the best split is finally applied. Then, this procedure is applied repeatedly until the data set is partitioned into k clusters. The algorithm starts like a "divisive hierarchical clustering" method, taking the entire set as the initial cluster, which is then recursively split into smaller clusters (Steinbach et al., 2000b, p.8f). The basic bisecting k-means algorithm is outlined in algorithm 5.3 in pseudo-code. The algorithm can be stated in general terms, as shown here, imposing only few constraints on the formal representation of the investigated data. A criterion needs to be defined in an actual implementation for the selection of a particular cluster that should be split. A second criterion needs to be defined to assign scores to the I preliminary splits produced by the k-means algorithm. However, k-means itself poses more specific requirements on

Algorithm 5.3 Basic Bisecting k-means algorithm

1: Let $D = \{x_1, \ldots, x_N\}$ be a set of N data items
2: Let q be the number of desired clusters
3: Define D as the initial cluster, containing all data items
4: **repeat**
5: Select a cluster to split
6: **for** I iterations **do**
7: Split the selected cluster using basic k-means with $k = 2$
8: **end for**
9: Select the best split and apply it to the selected cluster
10: **until** the number of q clusters is reached

the data representation (e.g. the necessary definition of a distance metric on the data items). The source code of the MATLAB implementation for the present experiment is listed in B.42 on page 347. It follows the definitions by Steinbach et al. (2000a, p.8f) and Steinbach et al. (2000b, p.1–2). The internally applied k-means clustering is done with the `kmeans` function provided by MATLAB.

Clustering is applied to the sample sets inside a `parfor`-loop which runs for 1000 iterations (in the main function `experiment1`). The `parfor` construct is a loop which runs its individual iterations in a number of parallel processes. This speed optimisation is possible since each individual clustering run is independent from any other iteration.

Evaluation

The results of each individual clustering iteration are evaluated separately and collected. First, the function `assignClassesToClusters`, shown in listing B.44, creates a table where each cell c_{ij} contains the number of frames of the i-th cluster which belong to the j-th reference class. Based on this table, it is possible to compute the *adjusted Rand index*, *purity* and *V-measure* according to equations 5.10, 5.12 and 5.13, respectively. The implementation of purity is shown in source code listing B.46. It follows the definition of Manning et al. (2009, p.356f). The implementation of V-measure, shown in listing B.47, follows Rosenberg and Hirschberg (2007, p.410f). The adjusted Rand index is implemented according to equation 5.10 on page 173 following Hubert and Arabie (1985, p.198). The source code is listed in B.45.

The number of classes in the present experiment corresponds to the number of reference labels. This depends on the decision, which labels from which annotation layer should be used for evaluation. Note that annotation of continuous data like a speech signal with a discrete set of labels always contains some arbitrary decisions, like, for example, labelling a plosive with a single label or labelling the silence and release parts of the phone with separate labels. In the present experiment the annotated phonetic

segments are taken as the reference. Thus, $C = \{\mathrm{p}, \mathrm{r}, \mathrm{a}, \mathrm{l}, \mathrm{k}, \mathrm{i}, \mathrm{ɨ}, \mathrm{u}\}$ corresponds to the set of phone labels and the number of generated clusters is equal to the number of reference classes, i.e. $|\Omega| = |C| = 8$.

In order to evaluate the results with a confusion matrix (cf. section 3.6.2), another table is created which takes the same approach as the purity measure and matches each cluster to its *majority class*, i.e. to the class which is most frequent in the cluster. The confusion matrix is then arranged in the usual way with clusters represented by rows and classes by columns such that the matching data lies on the main diagonal. Computing this confusion matrix based on a clustering solution is a non-trivial optimisation problem. If no two clusters have the same majority class (i.e. if no two rows have their maximum value in the same column), the rows of the contingency table can be sorted such that the cells with the maximum value for each cluster lie on the main diagonal. This way, the clusters can trivially be matched to different reference classes. However, if two or more clusters share their matching majority class because most of their items belong to the same reference class, the construction of an optimal confusion matrix becomes a non-trivial problem. Two approaches to the construction of a (pseudo-) confusion matrix are employed in this experiment: (1) a "cluster sorting" method and (2) a "cluster matching" method.

The "cluster sorting" method gives a simple heuristic approximation to an optimal cluster-to-class assignment. First, the majority class is computed for each cluster. This constitutes the primary sort key. To approximately solve conflicts, the following heuristic is applied: The secondary sort key for each cluster is constructed according to the column distance of their second most frequent class from their majority class, and the distance of the third most frequent class etc. See source code in listing B.38 at the part commented with: "compute sorted confusion matrix". The resulting matrix preserves the number of generated clusters but introduces distortions for cases where multiple clusters share a majority class. Depending on how these clusters are sorted, the confusion matrix can change considerably. No attempt was made to obtain an optimal solution for this problem in the present experiments. The matrices constructed by this method provide an approximate visualisation of the total confusion over all clusters.

The "cluster matching" method, on the other hand, has the effect that the resulting confusion matrix can contain all-zero rows. When two clusters are matched with the same class, another class will necessarily remain unmatched. Its corresponding row in the table will then contain only zero values. The matrices constructed by this method provide a visualisation of the cluster purity. They can be used to evaluate cluster internal confusion, while the relations between clusters appear distorted.

The confusion matrices are computed separately for each clustering iteration and added to an overall confusion matrix. Two graphs are produced for the overall confusion matrices: one showing the absolute values in the tables and another one showing the relative class distributions for each cluster. The function `confusionEvaluation` shown

in source code listing B.48 computes the *classification error* (cf. 5.2.4) based on the overall confusion matrices and prints statistics about the cluster sizes. Additionally, a "total cluster size error" is computed according to the following formula:

$$\text{cluster size error} = \frac{1}{|\Omega|} \sum_{i=1}^{|\Omega|} \frac{|\,(|\omega_i| - |c_i|)\,|}{|c_i|} \quad , \tag{5.20}$$

where equal indices for clusters and classes indicate that a cluster ω_i is matched with a class c_i. This is an *ad hoc* measure to obtain an indication for the clustering method's ability to create clusters with the correct size and how this relates to the different phone classes.

5.3.3. Results

The quality of the clustering results is measured with the *adjusted Rand index*, *purity* and *V-measure*[21].

Table 5.2 shows average purity, V-measure and adjusted Rand index over the 1000-fold clustering. The results are shown for all speakers and all three data types separately. The standard deviation (according to the MATLAB function `std`) is given for each mean value in brackets. Column "audio" shows the results for the acoustic data, column "EMA" the results for articulatory data, and column "EMA+audio" shows the results for the combined data type.

A comparison of purity for the three different data types shows that the clustering performed best on articulatory data for speakers 2 and 3. For speaker 1, articulatory data performs worst and the best performance can be observed for the combined data type. The same observation can be made for V-measure and the adjusted Rand index. Comparing the data types it can also be observed, that adding articulatory data improves the performance over acoustic data alone, i.e. the combined data yields better results than the acoustic data. Starting from articulatory data, however, the results are degraded with the change to the combined data for speakers 2 and 3.

Cluster sizes

The average cluster sizes shown in table 5.3 on page 188 are based on the "sorted" confusion matrices[22]. The table shows the statistics for all three speakers and all three

[21] See section 5.2.4 for a discussion of cluster evaluation measures.

[22] In this case, the underlying depiction of the clustering results as a confusion matrix has no effect on the cluster size statistics. The sorted tables serve merely as a temporary storage from which the cluster size statistics are computed at the end of the experiment in function `confusionEvaluation` shown in Listing B.48. Although the cluster size statistics can also be computed for the "matched" confusion tables (as can be seen in the source code), they would not be valid due to the distortions introduced by

Purity

speaker	audio	EMA	EMA+audio
1	0.430 (0.018)	0.376 (0.023)	0.434 (0.018)
2	0.425 (0.014)	0.632 (0.031)	0.508 (0.040)
3	0.474 (0.024)	0.578 (0.031)	0.478 (0.024)

V-measure

speaker	audio	EMA	EMA+audio
1	0.320 (0.014)	0.270 (0.020)	0.322 (0.015)
2	0.342 (0.013)	0.574 (0.022)	0.402 (0.031)
3	0.379 (0.022)	0.554 (0.021)	0.382 (0.021)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	0.177 (0.011)	0.153 (0.017)	0.180 (0.014)
2	0.189 (0.017)	0.419 (0.031)	0.261 (0.039)
3	0.239 (0.017)	0.387 (0.029)	0.242 (0.017)

Table 5.2.: Experiment 1: Average clustering quality of all clusterings per speaker. Top: purity; middle: V-measure; bottom: adjusted Rand index values — values in brackets denote standard deviations.

data types. The figures show that in terms of the size of the generated clusters, the clustering procedure performs best on articulatory data (column "EMA") where the average cluster sizes are closest to the real class sizes. The total size error on EMA data is 0.016 for speaker 1, 0.057 for speaker 2 and 0.014 for speaker 3. On acoustic data the respective errors are 0.155, 0.153 and 0.064 for speakers 1, 2 and 3, respectively. On the combined articulatory and acoustic data, the total cluster size errors are 0.108, 0.113 and 0.082 for the three speakers. This indicates that the clustering method generates clusters of appropriate sizes most often with the articulatory data. Comparing the different clusters, it can be seen that the consonants at the top of the tables tend to be in smaller clusters while the vowels at the bottom of the tables tend to be in larger clusters for the acoustic data and (to a lesser degree) the combined data type.

matching with the clusters' majority classes.

Speaker 1 (class size: 110)

cluster	audio	EMA	audio+EMA
k-clusters	80.324 (43.821)	110.406 (29.777)	86.022 (51.633)
p-clusters	83.472 (53.976)	110.162 (31.203)	86.268 (57.012)
r-clusters	86.080 (56.980)	108.032 (30.897)	98.606 (63.371)
l-clusters	98.859 (74.141)	109.545 (32.803)	103.986 (77.324)
u-clusters	124.503 (104.877)	105.702 (30.274)	112.356 (94.743)
ɨ-clusters	120.521 (108.438)	111.751 (31.707)	121.356 (103.203)
i-clusters	117.336 (89.719)	114.352 (30.331)	116.641 (86.650)
a-clusters	122.514 (102.641)	110.419 (31.349)	119.598 (100.049)
total	104.201 (79.324)	110.046 (31.043)	105.573 (79.248)

Speaker 2 (class size: 116)

cluster	audio	EMA	audio+EMA
k-cluster	89.951 (65.583)	128.359 (39.837)	96.462 (41.267)
p-cluster	86.500 (59.539)	117.985 (39.736)	104.785 (48.275)
r-cluster	101.910 (60.650)	114.732 (44.828)	107.880 (51.384)
l-cluster	100.063 (61.686)	111.960 (41.153)	118.278 (50.376)
u-cluster	109.799 (96.035)	110.420 (35.956)	100.529 (69.334)
ɨ-cluster	109.752 (98.076)	105.501 (40.034)	106.194 (73.878)
i-cluster	140.847 (99.639)	127.456 (40.648)	138.004 (75.146)
a-cluster	135.568 (104.668)	110.526 (45.421)	132.046 (77.603)
total	109.299 (80.734)	115.867 (40.952)	113.022 (60.908)

Speaker 3 (class size: 104)

cluster	audio	EMA	audio+EMA
k-cluster	96.035 (53.482)	105.021 (29.736)	95.879 (52.512)
p-cluster	99.290 (99.290)	102.222 (37.884)	99.573 (54.118)
r-cluster	102.480 (56.277)	104.882 (29.826)	103.842 (56.128)
l-cluster	97.649 (72.339)	102.866 (29.273)	97.942 (74.407)
u-cluster	94.507 (83.016)	103.350 (32.559)	94.759 (82.003)
ɨ-cluster	112.766 (99.452)	105.775 (33.582)	117.389 (98.914)
i-cluster	118.471 (98.157)	105.996 (30.269)	120.710 (96.702)
a-cluster	103.891 (98.607)	101.356 (29.960)	93.703 (87.848)
total	103.136 (77.427)	103.934 (31.960)	102.975 (75.329)

Table 5.3.: Experiment 1: Average cluster sizes and standard deviations (in brackets).

Confusion matrices

The adjusted Rand index, purity and the V-measure give mixed results for the three speakers. In order to investigate the clustering results in more detail, (pseudo-) confusion matrices have been computed.

The confusion matrices are visualised as *confusion graphs*. The values of the table cells are depicted by bar plots. The graphs combine all three data types which allows a faster visual comparison of the overall distributions in the confusion matrices. The top rows in figures 5.2, 5.3 and 5.4 show a visualisation of the matched confusion matrices and the bottom rows show a visualisation of the sorted confusion matrices. The left columns of the figures visualise the confusions' absolute values and the right columns show the relative distributions for each cluster. Dark grey bars on the left correspond to articulatory data, light grey bars on the right to acoustic data, and the medium grey bars in the middle correspond to the combined data type.

As pointed out above, the confusions based on the matched tables provide a visualisation of cluster purity. The figures show that /k/, /p/ and /r/ are most often confused, i.e. the respective clusters contain many data frames from the other classes. The vowel clusters show less confusions. The figures show an interesting trend: confusions are larger on acoustic data towards the top of the tables and larger for articulatory data towards the bottom of the tables. In other words, clustering of the consonants (esp. the obstruents) is easier with articulatory data, and clustering of the vowels (esp. the open vowel /a/) is easier with acoustic data.

5.3.4. Experiment 1b: Baseline system comparison

Intuitively, the results of the first experiment show only moderate performance of the clustering procedure in terms of classification accuracy and cluster quality. As pointed out in section 3.6, it is common practice to evaluate the performance of a system by comparing it against a set baseline. Such a comparison of the results against a baseline system will allow a more informed estimate of the quality of the results.

In this section, I present a comparison of the results of experiment 1 against a simple baseline system which is a random clustering in a kind of Monte-Carlo simulation. The outline of this experiment is identical to that of the previous experiment, as shown in algorithm 5.2 on page 181. Data, method and evaluation are the same as described in section 5.3 for experiment 1. The clustering function now is a random assignment of cluster labels to data items. The source code of this random clustering function is listed in B.52. The function takes the set of data frames D and a parameter k as input. It creates an equal number of cluster IDs in the range from 1 to k, and assigns them randomly to each frame in D. The remaining parameters are the same as for experiment 1. Listing B.51 on page 361 shows the main MATLAB script `clusteringExperiment1baseline` which defines the parameters and starts the experiment.

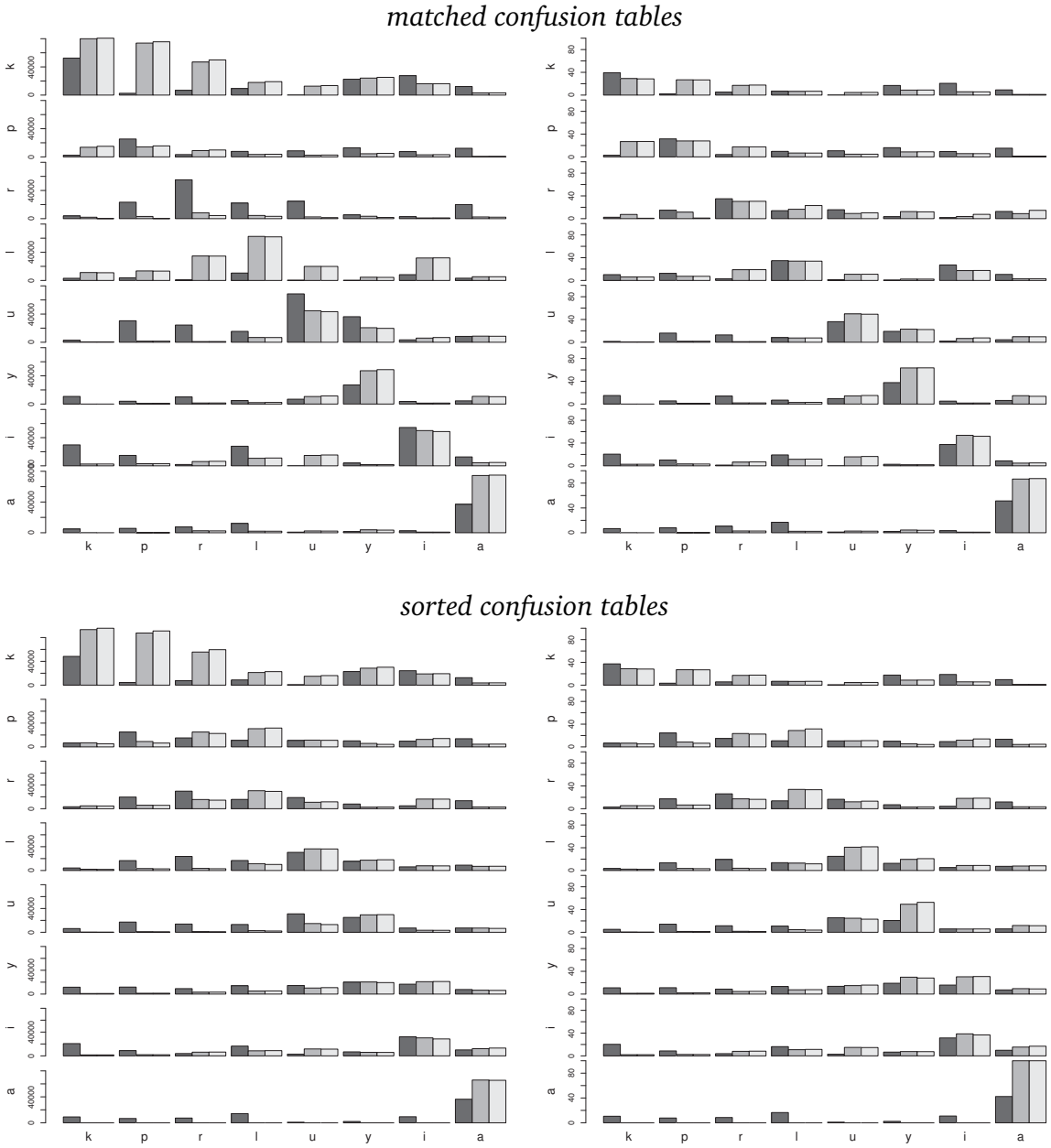


Figure 5.2.: Experiment 1: Confusion graphs for speaker 1. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

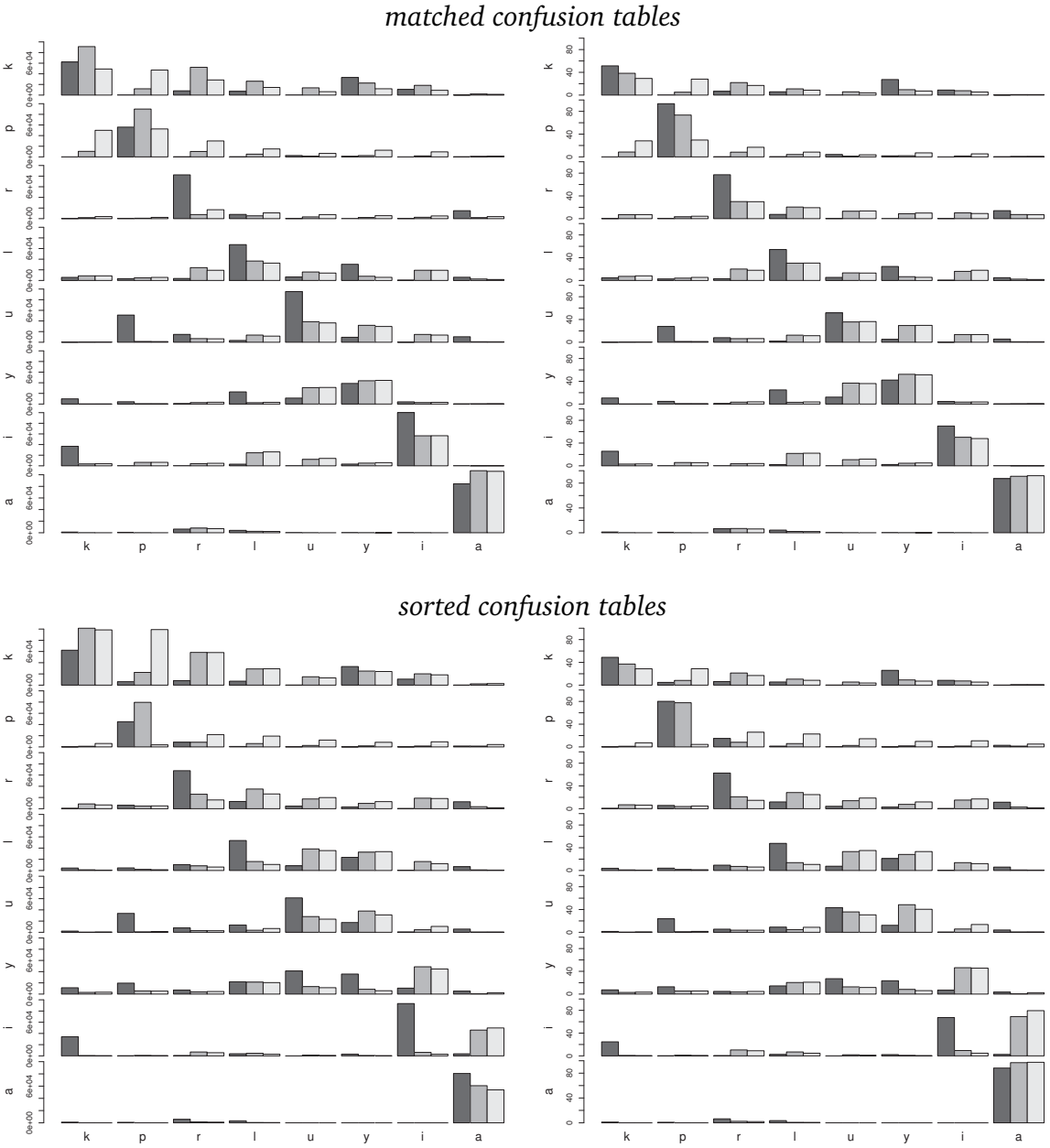


Figure 5.3.: Experiment 1: Confusion graphs for speaker 2. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

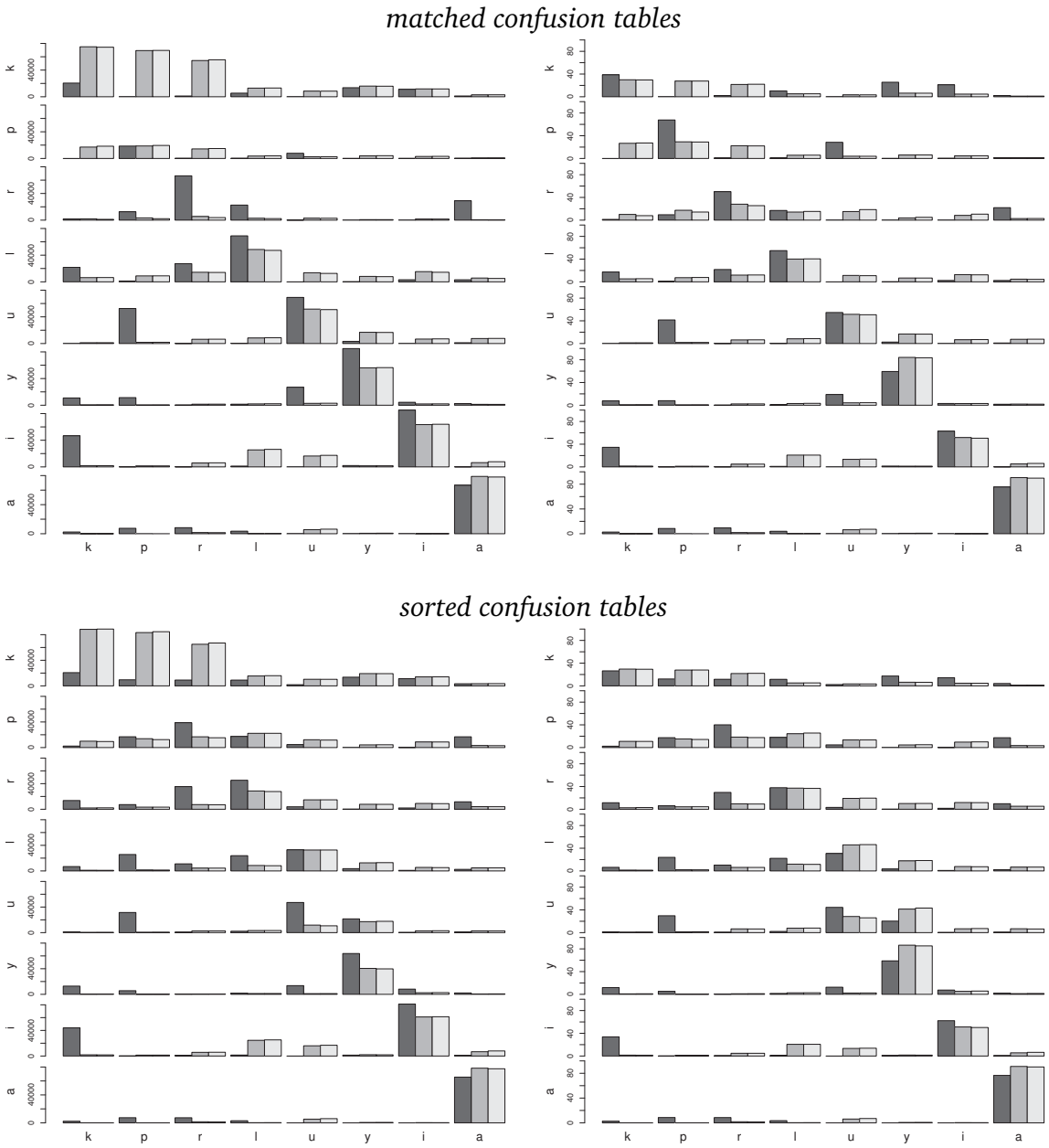


Figure 5.4.: Experiment 1: Confusion graphs for speaker 3. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

Purity

speaker	audio	EMA	EMA+audio
1	0.171 (0.006)	0.171 (0.006)	0.172 (0.007)
2	0.170 (0.006)	0.170 (0.006)	0.170 (0.006)
3	0.173 (0.007)	0.173 (0.006)	0.173 (0.007)

V-measure

speaker	audio	EMA	EMA+audio
1	0.014 (0.003)	0.014 (0.003)	0.014 (0.003)
2	0.013 (0.003)	0.013 (0.003)	0.013 (0.003)
3	0.015 (0.003)	0.014 (0.003)	0.014 (0.003)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	0.000 (0.002)	0.000 (0.002)	0.000 (0.002)
2	0.000 (0.001)	0.000 (0.001)	0.000 (0.001)
3	0.000 (0.002)	0.000 (0.002)	0.000 (0.002)

Table 5.4.: Experiment 1b: Average clustering quality of all random clusterings per speaker. Top: purity; middle: V-measure; bottom: adjusted Rand index values — values in brackets denote standard deviations.

Results of the baseline system

The average cluster quality in terms of the three employed quality measures *purity*, *V-measure* and *adjusted Rand index* is shown in table 5.4 on page 193. The results confirm the expected scores for random solutions. As expected, a comparison of the different data type conditions shows no significant differences with all evaluation measures for all speakers. A comparison between speakers shows no significant differences, either. These observations on the results shown in table 5.4 confirm the validity of the implementation of the method performing a random clustering on the given data sets. The comparison between experiment 1 and experiment 1b shows the largest absolute differences in the scores of the V-measure and the least absolute differences for the adjusted Rand index.

The primary purpose of experiment 1b is the comparison between its results and the results for experiment 1 shown in table 5.4 on page 193. All evaluation measures indicate considerably higher average cluster quality for experiment 1 than for the baseline. This

is true for all three speakers and all data types. Despite the apparently low scores for the clusterings in experiment 1, a comparison against a baseline with random cluster assignments reveals a meaningful behaviour. Although the clustering quality is far from perfect, the achieved partitions of the data seem to reflect some of the inherent linguistic information.

A confusion graph for speaker 1 is shown in figure 5.5. The graphs for the "matched" confusion tables show the distortion effect discussed above. The peaks are much more pronounced in comparison to the peaks in the graphs for the "sorted" confusion graphs. Overall, the distribution of classes over clusters is relatively flat. The graphs for speakers 2 and 3 are omitted. The clustering procedure does not depend on the data and therefore produces similar results for all speakers on all data types.

5.3.5. Experiment 1c: Refinement with contiguous frame sequences

One aspect of experiment 1 as presented in section 5.3 that could be seen as a weak point is the procedure generating the random sample sets. The purely random selection of the individual data frames from the corpus does not generate contiguous sequences. This could be problematic for discontinuous phones like the obstruents /p/ and /k/ in the Polish EMA corpus, or the trill /r/. Depending on the particular location within these segments, two frames from such a phone can be dramatically different. In an extension to experiment 1, I implemented a sampling procedure which generates random sample sets with contiguous stretches of frames taken from the corpus. The reasoning behind the usage of sequences of contiguous data frames is related to the "extended context method" in the speech segmentation experiments presented by Duran et al. (2010b, p.156f): context is often required to decide whether two individual frames of a speech signal belong to the same segment category. However, in contrast to the segmentation experiments with the "extended context method", immediately adjacent frames are selected here without consideration of any changes in the signals[23] . The procedure is still a frame-by-frame clustering. In contrast to experiment 1, the frames are selected mainly from the same local contexts.

The source code is listed in section B.2.3 in the appendix, starting on page 363. As for the baseline system presented in the previous section, this experiment shares most of the source code with the implementation of experiment 1. The initialisation script in listing B.53 defines the parameters and starts the main function that runs the clustering experiment. The sampling function is replaced by one which attempts to take frames in contiguous sequences from the corpus. The commented source code of the sampling function `getSampleSetCont` is shown in listing B.54. The sampling of the data set is

[23]Speech segmentation is discussed in detail in chapter 4, including the experiments by Duran et al. (2010b)

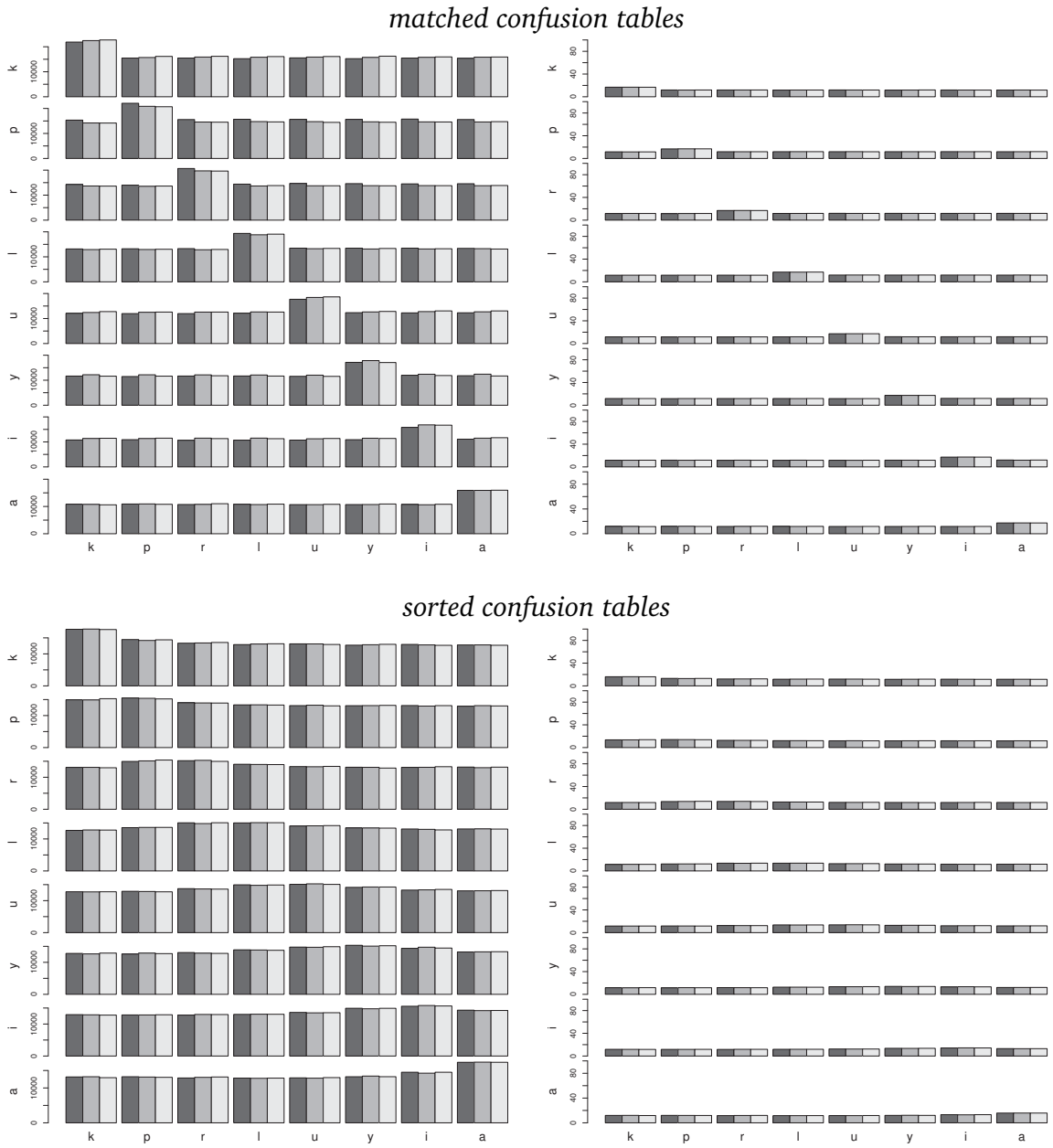


Figure 5.5.: Experiment 1b: Confusion graphs for speaker 1. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

Purity

speaker	audio	EMA	EMA+audio
1	0.516 (0.026)	0.434 (0.039)	0.526 (0.031)
2	0.496 (0.025)	0.680 (0.051)	0.588 (0.041)
3	0.556 (0.037)	0.602 (0.046)	0.568 (0.044)

V-measure

speaker	audio	EMA	EMA+audio
1	0.470 (0.027)	0.355 (0.038)	0.473 (0.028)
2	0.466 (0.024)	0.642 (0.039)	0.528 (0.033)
3	0.531 (0.036)	0.582 (0.033)	0.539 (0.038)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	0.310 (0.025)	0.205 (0.034)	0.316 (0.112)
2	0.302 (0.029)	0.486 (0.054)	0.390 (0.046)
3	0.369 (0.032)	0.410 (0.046)	0.381 (0.106)

Table 5.5.: Experiment 1c: Average clustering quality of all clusterings per speaker. Top: purity; middle: V-measure; bottom: adjusted Rand index values — values in brackets denote standard deviations.

based on a random selection. First, one frame is selected at random from a given class. Then a number of adjacent frames is added up to a maximum specified in the global setup. This maximum is set to 25 frames (which corresponds to stretches of 0.1 s). The remaining functions and parameters are the same as in experiment 1. The same evaluation scheme is applied.

Results on contiguous frame sets

The sample sets in this experiment contain more frames which are immediately adjacent to each other and, due to this fact, probably more similar to each other. It could be expected that a clustering procedure performs better on such kind of data in comparison to the sample sets created in experiment 1.

Table 5.5 on page 196 shows the results of experiment 1c with contiguous frame sets.

A comparison with the results for experiment 1 in table 5.2 on page 187 reveals an improvement of all measurements — accompanied by an increased standard deviation. The overall picture is, however, unchanged: The highest evaluation scores can be observed with articulatory data for speakers 2 and 3. Again, articulatory data shows the worst results for speaker 1. Comparing the three data types, it is again the case that the combined signals gets higher scores than the acoustic data alone.

Figures 5.6, 5.7 and 5.8 show the confusion graphs for this experiment for speakers 1, 2 and 3, respectively. These figures show the same overall picture as the confusion graphs for experiment 1: clustering of the consonants is more successful with articulatory data and clustering of the vowels is slightly more successful with acoustic data.

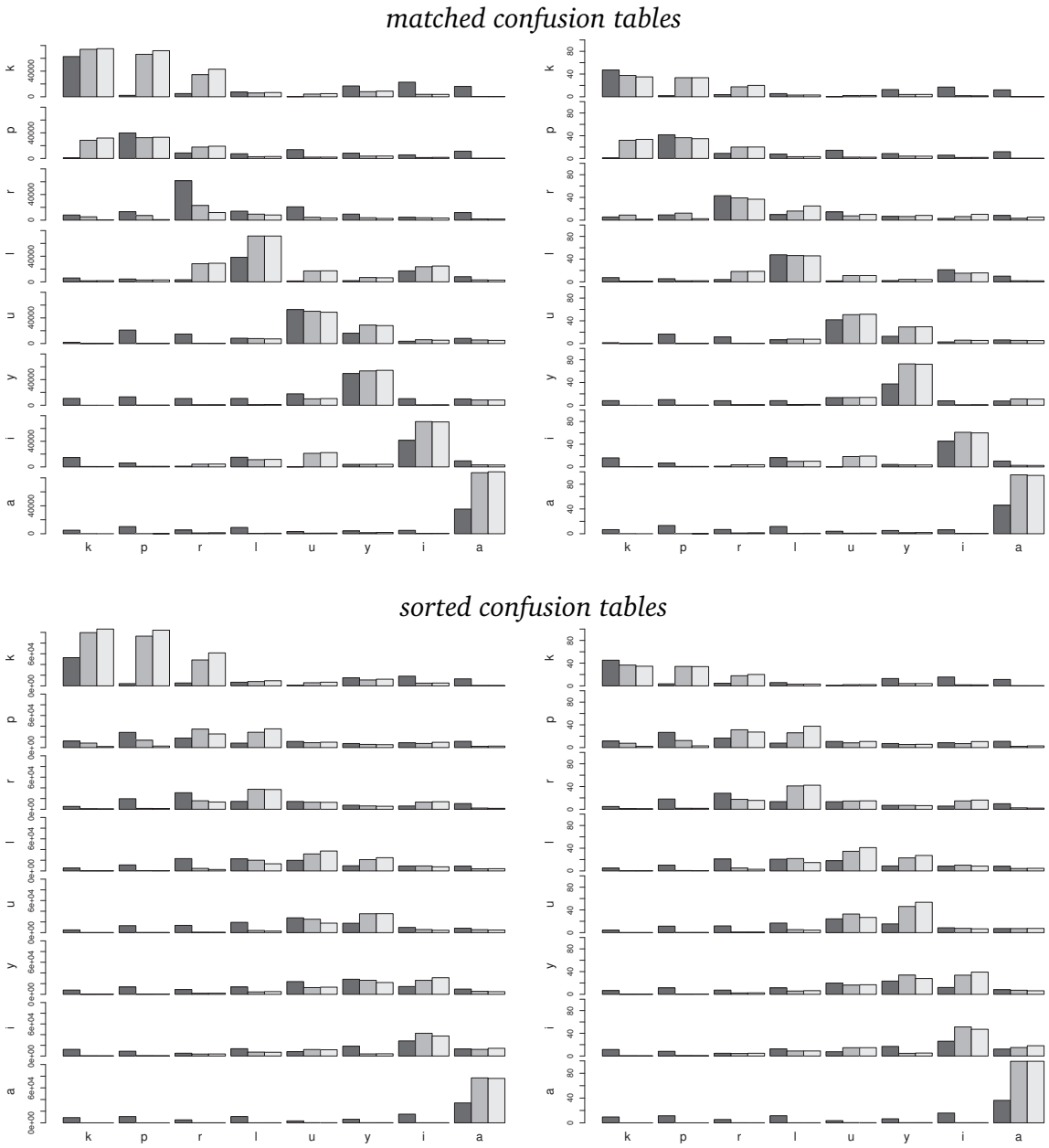


Figure 5.6.: Experiment 1c: Confusion graphs for speaker 1. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

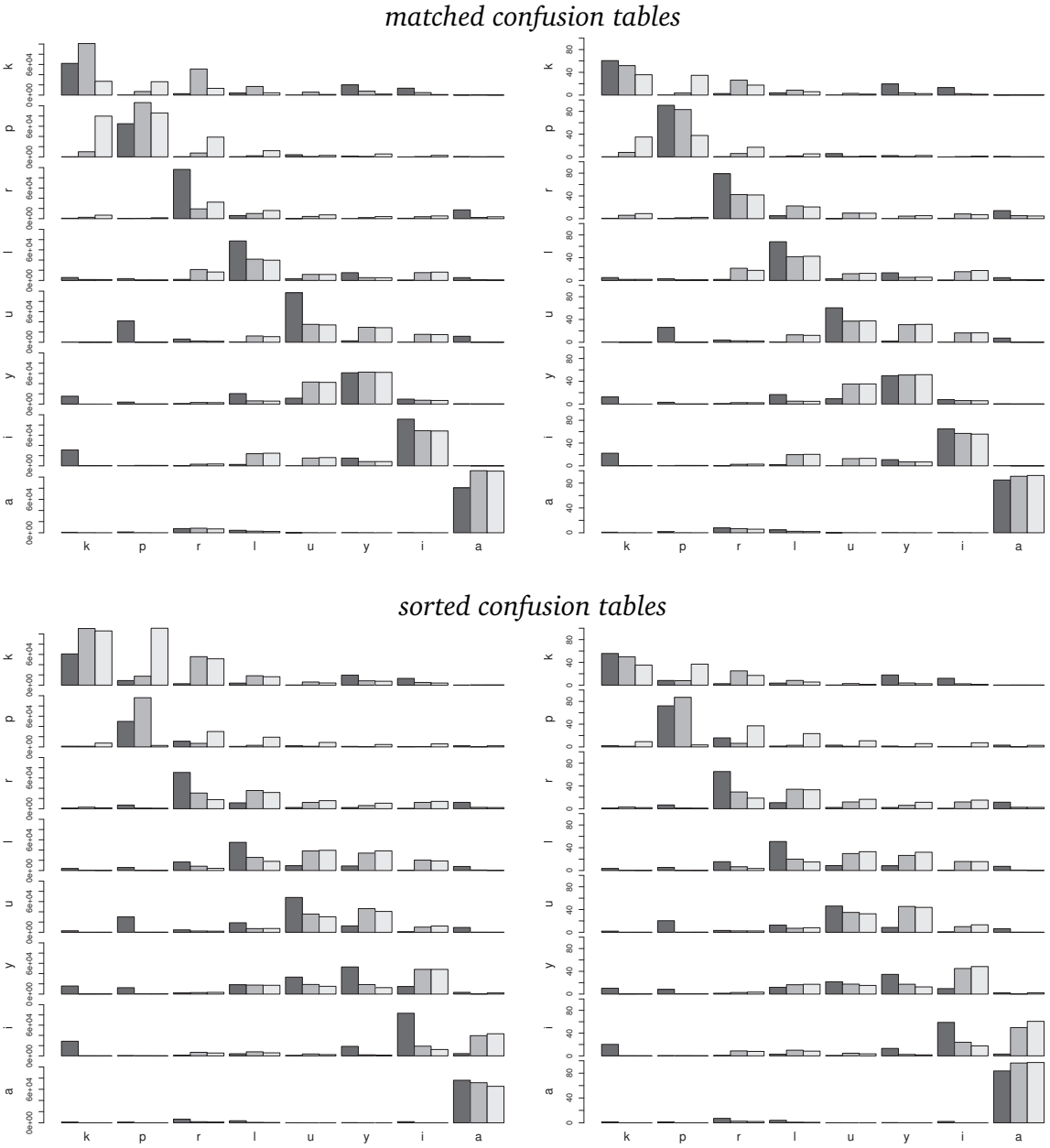


Figure 5.7.: Experiment 1c: Confusion graphs for speaker 2. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

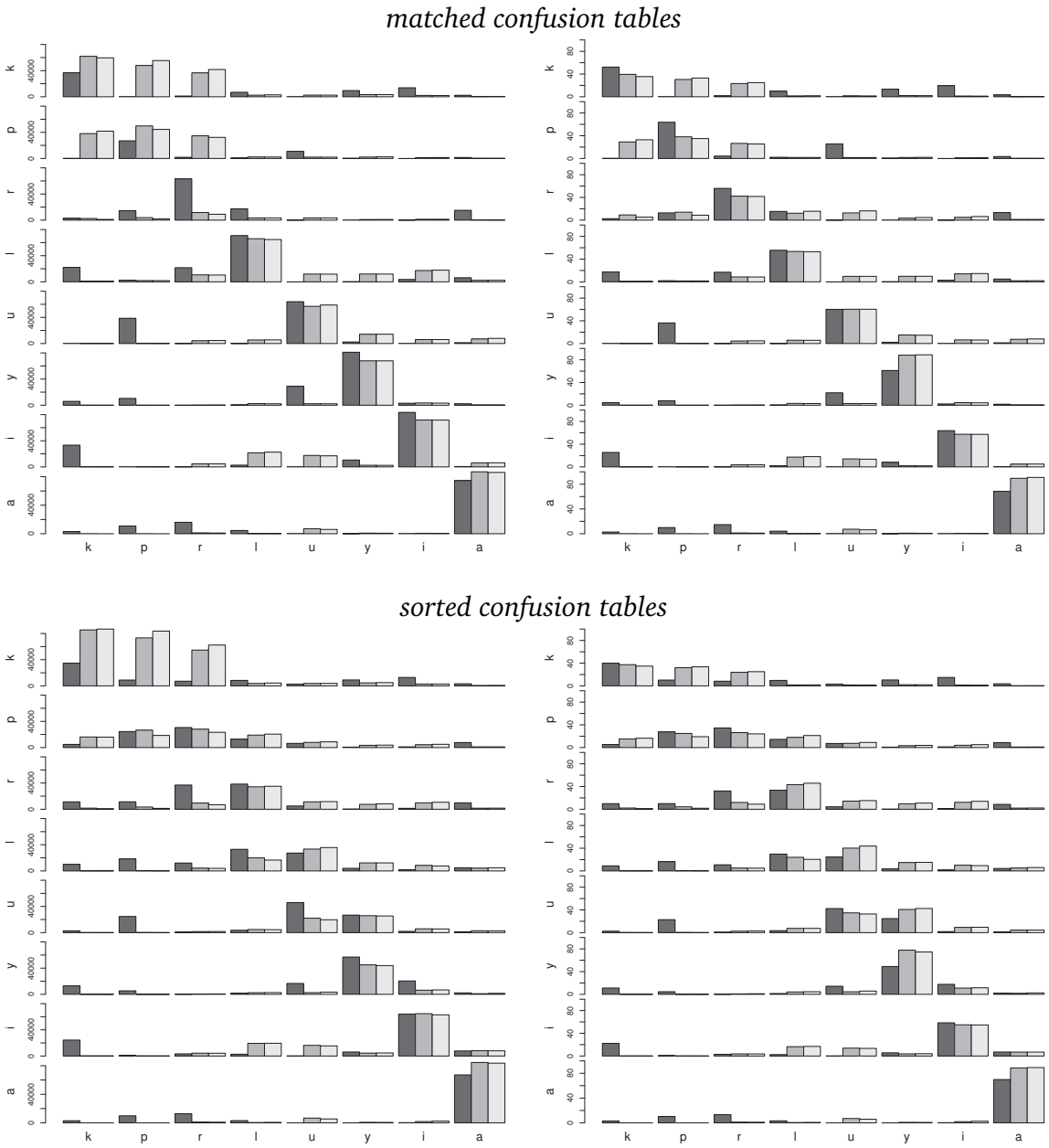


Figure 5.8.: Experiment 1c: Confusion graphs for speaker 3. The top row shows a visualisation of the "matched confusion matrices" and the bottom row shows a visualisation of the "sorted confusion matrices". The left column visualises the confusions' absolute values and the right column shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars correspond to acoustic data, and the medium grey bars correspond to the combined data type.

5.4. Experiment 2: Clustering Polish vowel segments

This section presents an experiment which employs a clustering procedure to investigate the within-category discriminability of acoustic and articulatory speech data. The experiment has been presented previously at the LabPhon conference by Duran et al. (2012b).

The motivation for the experiment presented in this section is based on the observation that Polish obstruent–sonorant clusters show differences in voicing behaviour. Word-initial sonorants preceded by a voiceless obstruent in CCV clusters tend to be voiced almost through their entire duration, with only slight initial devoicing. In word-final positions, however, sonorants in a similar environment tend to be fully devoiced (Bruni, 2011, p.118f). Phonologically, this behaviour is attributed to *extrametricality*. Word-final sonorants are not part of the syllable structure and are therefore not licensed for [VOICE] (Gussmann, 1992, p.45f). This reasoning is supported by the patterns in coordination of the articulatory gestural timing within Polish syllables. Browman and Goldstein (1988, p.96) investigated articulatory relations of English consonants in clusters and reported articulatory stability of the consonantal distances with regards to the vowel in word-onset CCV clusters (the so called *C-centre effect*) and its lack in their word-final counterparts. Mücke et al. (2010) show that obstruent–sonorant clusters exhibit similar behaviour. Word-initial obstruents in words like ⟨pranie⟩ 'laundry' shift leftwards, whereas the following sonorant shifts rightwards in order to maintain stable consonantal distance relative to the vocalic target. Consonants in word-final clusters, however, like in ⟨Cypr ⟩ 'Cyprus' show no such patterning, where only the sonorant shifts rightwards, while the preceding obstruent demonstrates no leftward shift. If that is indeed the case in Polish, one would expect to see only little if any acoustic or articulatory anticipation of the word-final sonorant in the preceding vowel nucleus. The experiment presented in this section thus investigates whether an influence of the word-final sonorant is visible. For this purpose a clustering methodology is applied similar to the ones described in the preceding section.

In the previous experiment, a frame-by-frame clustering is applied with reference classes that correspond to phone segment types. The reference classes in this experiment correspond to context-sensitive segment labels. Acoustic and articulatory data is taken from vowel segments which occur in VC and VCC contexts. According to these contexts, the data points from the vowels are divided into two reference classes. The previous study is based on the assumption that the reference labels are indeed valid and the clustering procedure is applied to investigate the amount of category information available in the different data representations. In contrast to that, the reasoning of this experiment is different. The class labels constitute the tested hypothesis. It is assumed that if the clustering procedure succeeds in discriminating the data (which is taken from segments with the same phonetic label) according to the reference classes this would support the systematic distinction according to the vowel's context. Co-articulatory phenomena

	class		class
speaker	ik	ikr	size
1	116	145	87
2	114	145	86
3	110	143	83

	class		class
speaker	iC	iCC	size
1	441	145	109
2	447	145	109
3	512	143	107

	class		class
speaker	ɨp	ɨpr	size
1	98	87	65
2	118	107	80
3	131	136	98

	class		class
speaker	ɨC	ɨCC	size
1	98	180	74
2	118	222	89
3	131	244	98

Table 5.6.: Experiment 2: Number of frames per reference class

have been extensively investigated in phonetics and speech processing research. This present experiment is focused in particular on the (absence of) the C-centre effect in Polish obstruent–sonorant codas. The possible effect of a present word final sonorant is investigated by testing whether a frame-by-frame clustering procedure is able to separate vowels from the same phone class accordingly.

5.4.1. Speech material

The speech material is taken from the Polish EMA corpus (described in section A.1 on page 251). The data representation is the same as in experiment 1 (section 5.3). Only data from the labelled vowel segments is used, taken from the 'emphasis' part of the corpus for all three speakers. The target vowels occur in syllables which are all word final and which are always followed by the same word ⟨aktualnie⟩ of the carrier phrase. Thus, the right contexts for each vowel used in this experiment are: VC(C)#a, where # indicates a word boundary. The same corpus pre-processing and data representation is used here as in experiment 1. Articulatory and acoustic data is taken from the vowel segments /i/ and /ɨ/ (as there is sufficient data available in the corpus only for these phones).

5.4.2. Method and implementation

The MATLAB implementation is based on experiment 1 as described in the previous section, and most of the source code is re-used for this experiment. The commented source code is listed in appendix B.2.4 starting on page 367. Only the scripts and functions which are different from the ones used in experiment 1 are listed (the remaining code can be found in section B.2.1).

The main script `clusteringExperiment2` is shown in B.55 on page 367. The script `clusteringExperiment2C` shown in B.56 on page 369 defines the parameters of the experiment with the broad C and CC classes (see below). As for the previous experiments, this script defines all parameters and starts the experiment. As can be seen, the experiment starts function `experiment1` which is also used in experiment 1. The basic difference between this experiment and the previous ones is the definition of the data set and its reference classes. This is implemented in function `getVowelClusteringDataFromPolishEMA`. Its source code is shown in B.57 on page 371.

Several frame-by-frame clustering experiments are evaluated which use the implementation of the bisecting k-means method as described in section 5.3.2. For the purpose of evaluation, a label is assigned to each of the classes according to the VC or VCC syllable context of a given vowel segment (e.g. "ɨpr" for the class of all frames of a /ɨ/ segment which is followed by the segments /pr/). As in experiment 1, the manually created annotation is not used for the clustering itself.

To avoid negative effects on the clustering caused by unequal sizes of the reference class sets, a random sampling procedure is applied to create corpus samples with equally-sized classes. To further account for effects caused by the random selection of frames from the original data, this sampling procedure is repeated 1000 times for every parameter combination. This is the same procedure as in experiment 1. Table 5.6 shows the number of data frames for each reference class in the three speaker sub-corpora and the corresponding class sizes (as in experiment 1, the class size corresponds to 75% of the size of the smallest set). The results reported here are averaged over the 1000-fold clusterings with the following experimental parameter settings:

2.1 Vowel: /i/; classes: $C = \{$"ik", "ikr"$\}$

This corresponds to the set of frames from /i/ segments which are followed by either a simple coda consisting of a /k/ segment or by a complex syllable coda consisting of a /kr/ cluster.

2.2 Vowel: /ɨ/; classes: $C = \{$"ɨp", "ɨpr"$\}$

This corresponds to the set of frames from /ɨ/ segments which are followed by either a simple coda consisting of a /p/ segment or by a complex syllable coda consisting of a /pr/ cluster.

2.3 Vowel: /i/; classes: $C = \{$"iC", "iCC"$\}$

This corresponds to the set of frames from /i/ segments which are followed by either a simple coda consisting of either a /k/, /l/ or /r/ segment or by a complex syllable coda consisting of a /kr/ cluster.

2.4 Vowel: /ɨ/; classes: $C = \{$"ɨC", "ɨCC"$\}$

This corresponds to the set of frames from /ɨ/ segments which are followed by either a simple coda consisting of a /p/ segment or by a complex syllable coda consisting of either a /pr/ or /kl/ cluster.

Experiments 2.3 and 2.4 define reference classes based on broad C and CC classes. The function `getVowelClusteringDataFromPolishEMA` refers to a function handle defined in `setup.classGetterFH` to determine the reference class for a given segment, if it is a target for clustering. In experiments 2.1 and 2.2, the function associated with the handle is `getTargetClass`, shown in source code listing B.58. In experiments 2.3 and 2.4, the respective function is `getTargetClassC`, shown in source code listing B.59.

5.4.3. Results

The same evaluation scheme is applied as in experiment 1. Three cluster quality measures are computed for each individual clustering result, accumulated over all 1000 repetitions and then averaged. The measures are the same as the ones used in experiment 1: *purity*, *V-measure* and the *adjusted Rand index*.

Purity, V-measure and the adjusted Rand index for experiments 2.1–2.4 are shown in tables 5.7 and 5.8 on pages 205 and 206, respectively.

The results of experiment 2.1 in the left part of table 5.7 show the highest scores for articulatory data (column "EMA"), and the same performance for the acoustic data and the combination of acoustic and articulatory data. The results for speaker 2 on articulatory data are considerably worse as compared to the results of the other two speakers. The same pattern is visible in the results of experiment 2.2 in the right part of table 5.7.

The results of experiment 2.3 in the left part of table 5.8 and the results of experiment 2.4 in right part of table 5.8 show the same overall pattern for speakers 1 and 3: the best scores are achieved with all evaluation measures for the articulatory data. The best scores for speaker 1 can be observed for acoustic data in table 5.8 for experiment 2.3, although only by a small margin. For experiment 2.4, the results for speaker 1 are again similar to those for the other two speakers.

Confusion graphs for experiment 2.1 are shown in figure 5.9 on page 208. The graphs are based on "matched" confusion tables (which are generated as described in section 5.3). The graphs show relatively little confusion on the articulatory data

2.1: $C = \{$“ik”, “ikr”$\}$

Purity

speaker	audio	EMA	EMA+audio
1	0.518 (0.013)	0.893 (0.139)	0.518 (0.014)
2	0.538 (0.021)	0.768 (0.020)	0.537 (0.021)
3	0.535 (0.019)	0.999 (0.002)	0.534 (0.019)

V-measure

speaker	audio	EMA	EMA+audio
1	0.002 (0.002)	0.628 (0.291)	0.002 (0.002)
2	0.006 (0.006)	0.371 (0.030)	0.006 (0.006)
3	0.006 (0.006)	0.991 (0.019)	0.006 (0.006)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	-0.003 (0.003)	0.629 (0.321)	-0.003 (0.003)
2	0.002 (0.007)	0.285 (0.042)	0.002 (0.007)
3	0.001 (0.006)	0.995 (0.010)	0.001 (0.006)

2.2: $C = \{$“ɨp”, “ɨpr”$\}$

Purity

speaker	audio	EMA	EMA+audio
1	0.536 (0.022)	0.998 (0.023)	0.535 (0.023)
2	0.522 (0.016)	0.535 (0.021)	0.523 (0.016)
3	0.517 (0.013)	0.718 (0.112)	0.518 (0.013)

V-measure

speaker	audio	EMA	EMA+audio
1	0.005 (0.005)	0.995 (0.050)	0.005 (0.006)
2	0.003 (0.003)	0.005 (0.005)	0.003 (0.003)
3	0.001 (0.002)	0.285 (0.172)	0.001 (0.002)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	-0.001 (0.007)	0.996 (0.055)	-0.001 (0.008)
2	-0.002 (0.004)	0.001 (0.007)	-0.002 (0.004)
3	-0.003 (0.003)	0.236 (0.145)	-0.003 (0.003)

Table 5.7.: Experiments 2.1 and 2.2: Average clustering quality of all clusterings per speaker and data type. Values in brackets denote standard deviations.

2.3: $C = \{$“iC”, “iCC”$\}$

Purity

speaker	audio	EMA	EMA+audio
1	0.522 (0.017)	0.687 (0.075)	0.521 (0.016)
2	0.611 (0.025)	0.596 (0.030)	0.610 (0.024)
3	0.544 (0.028)	0.953 (0.026)	0.543 (0.027)

V-measure

speaker	audio	EMA	EMA+audio
1	0.002 (0.003)	0.168 (0.098)	0.002 (0.005)
2	0.039 (0.017)	0.033 (0.026)	0.038 (0.016)
3	0.029 (0.048)	0.777 (0.071)	0.028 (0.046)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	-0.001 (0.004)	0.159 (0.106)	-0.002 (0.004)
2	0.047 (0.023)	0.037 (0.030)	0.046 (0.021)
3	0.008 (0.013)	0.823 (0.074)	0.007 (0.013)

2.4: $C = \{$“ɨC”, “ɨCC”$\}$

Purity

speaker	audio	EMA	EMA+audio
1	0.528 (0.020)	0.965 (0.033)	0.527 (0.020)
2	0.519 (0.014)	0.716 (0.086)	0.519 (0.014)
3	0.523 (0.017)	0.609 (0.025)	0.524 (0.017)

V-measure

speaker	audio	EMA	EMA+audio
1	0.004 (0.005)	0.826 (0.085)	0.003 (0.004)
2	0.002 (0.003)	0.185 (0.138)	0.002 (0.003)
3	0.003 (0.004)	0.046 (0.020)	0.003 (0.004)

Adjusted Rand index

speaker	audio	EMA	EMA+audio
1	-0.002 (0.006)	0.870 (0.084)	-0.002 (0.006)
2	-0.003 (0.003)	0.212 (0.162)	-0.003 (0.003)
3	-0.002 (0.005)	0.046 (0.023)	-0.001 (0.004)

Table 5.8.: Experiments 2.3 and 2.4: Average clustering quality of all clusterings per speaker and data type. Values in brackets denote standard deviations.

Experiment 2.1 baseline ($C = \{$"ik", "ikr"$\}$)

speaker	purity	V-measure	adj. Rand index
1	0.531 (0.022)	0.004 (0.006)	0.000 (0.008)
2	0.530 (0.022)	0.004 (0.005)	0.000 (0.007)
3	0.530 (0.023)	0.004 (0.006)	0.000 (0.008)

Experiment 2.2 baseline ($C = \{$"ɨp", "ɨpr"$\}$)

speaker	purity	V-measure	adj. Rand index
1	0.535 (0.027)	0.006 (0.009)	0.000 (0.012)
2	0.531 (0.024)	0.004 (0.006)	0.000 (0.009)
3	0.528 (0.021)	0.003 (0.005)	0.000 (0.007)

Table 5.9.: Experiment 2: Average clustering quality for the random clustering baseline. Values in brackets denote standard deviations.

and an almost equal distribution of classes over clusters for the other two data types. The confusion graphs for experiment 2.2 are shown in figure 5.10 on page 209. The labels "yp" and "ypr" are used in the graphs to denote the classes "ɨp" and "ɨpr". Figures 5.11 and 5.12 on pages 210 and 210 show the confusion graphs for experiments 2.3 and 2.4 with the broad consonant contexts. The overall pattern is the same for all four experiments.

5.4.4. Baseline comparison

As in experiment 1, the results are compared against a simple baseline method. A random clustering method is applied to the data and the same evaluation scheme is used to compute the cluster quality for each parameter combination. The baseline results for *purity*, *V-measure* and the *adjusted Rand index* are shown in table 5.9 on page 207. The results are not shown separately for the different data type conditions as the clustering is independent from that parameter[24].

[24]Note also that the results shown in section 5.3.4 for the baseline of experiment 1 have shown no differences for the different data types, apart from random fluctuations.

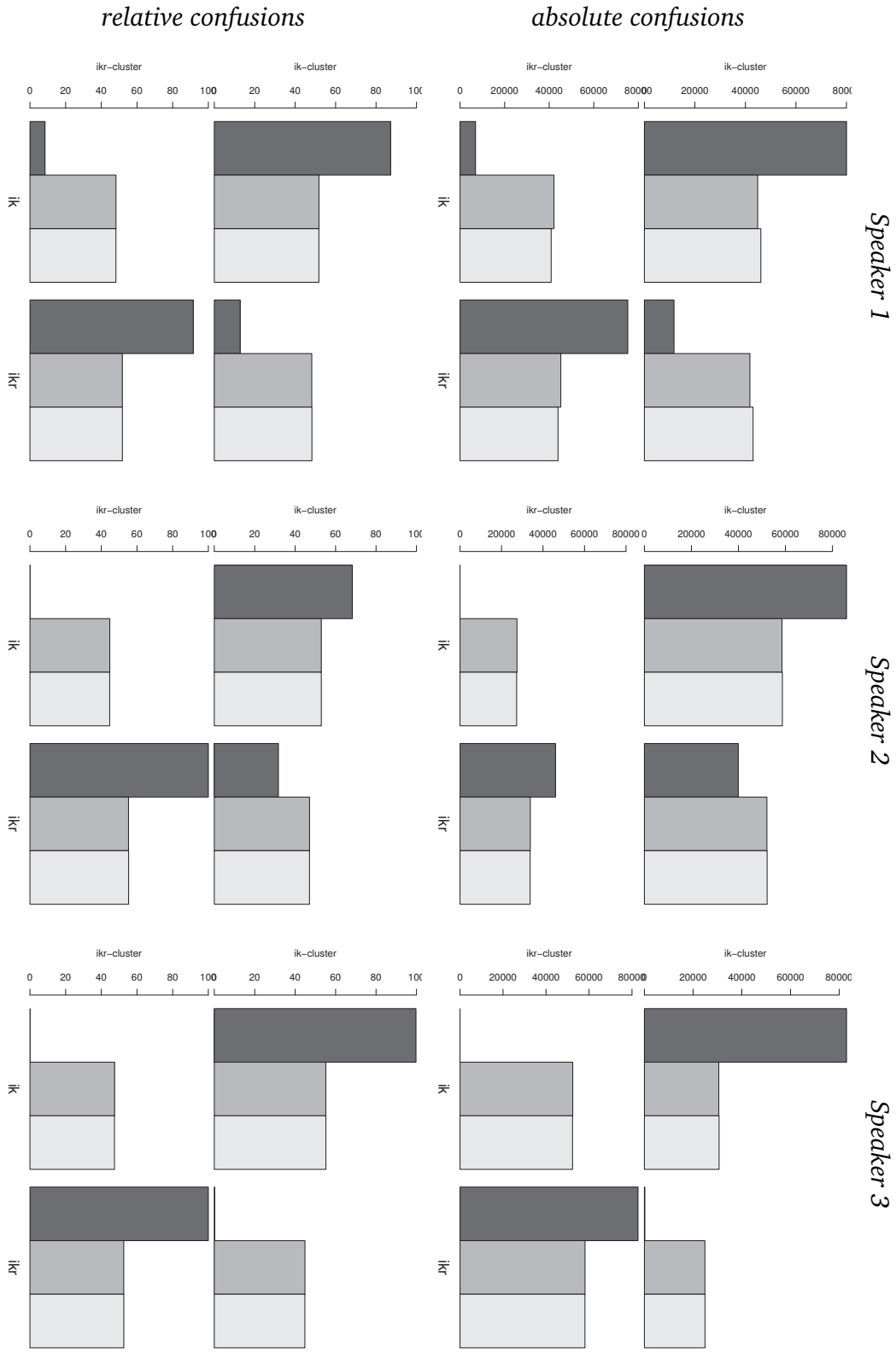


Figure 5.9.: Experiment 2.1: Confusion graphs on matched confusion tables. The top row visualises the confusions' absolute values and the bottom row shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars to acoustic data, and the medium grey bars to the combined data type.

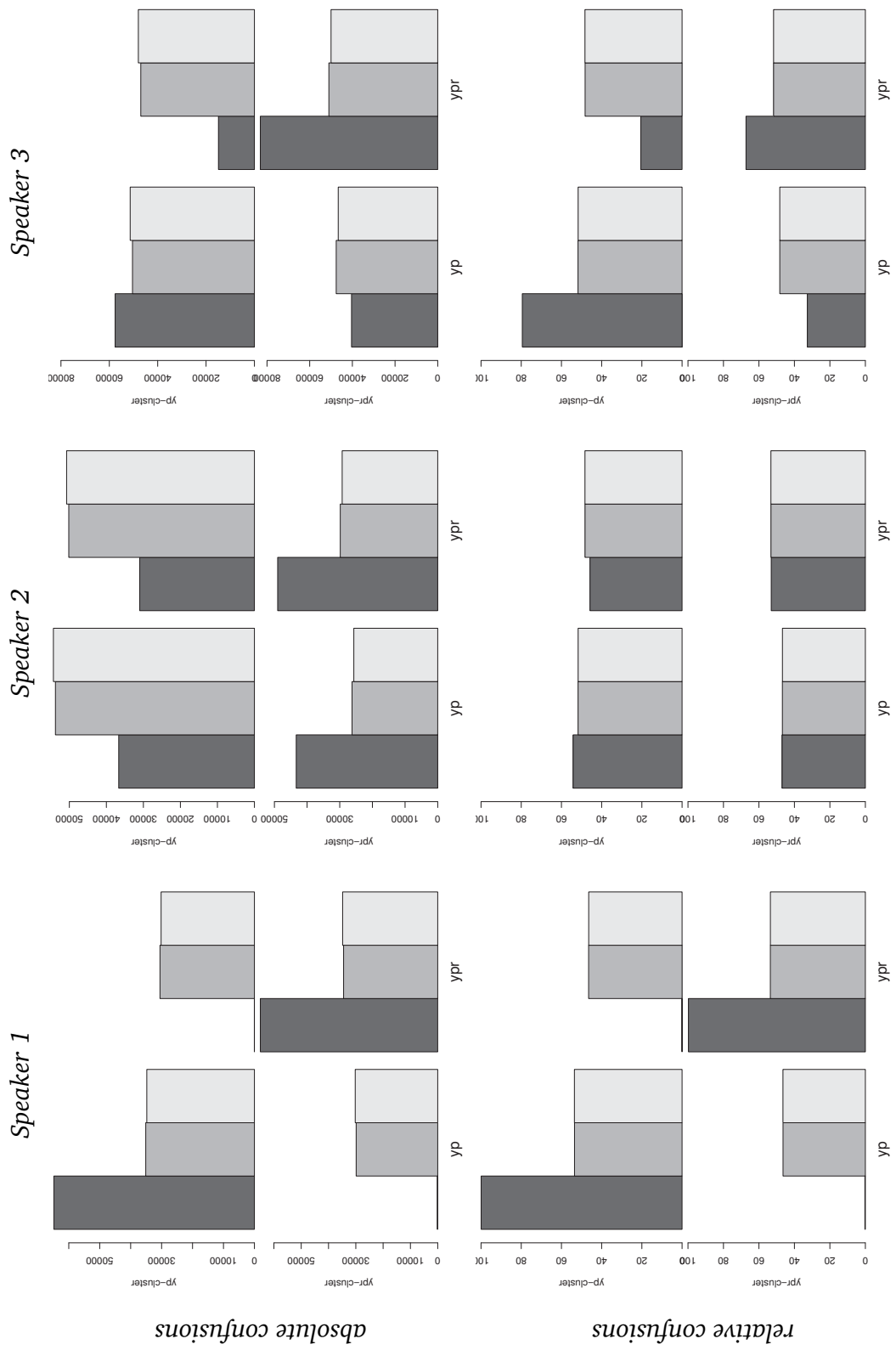


Figure 5.10.: Experiment 2.2: Confusion graphs on matched confusion tables. The top row visualises the confusions' absolute values and the bottom row shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars to acoustic data, and the medium grey bars to the combined data type.

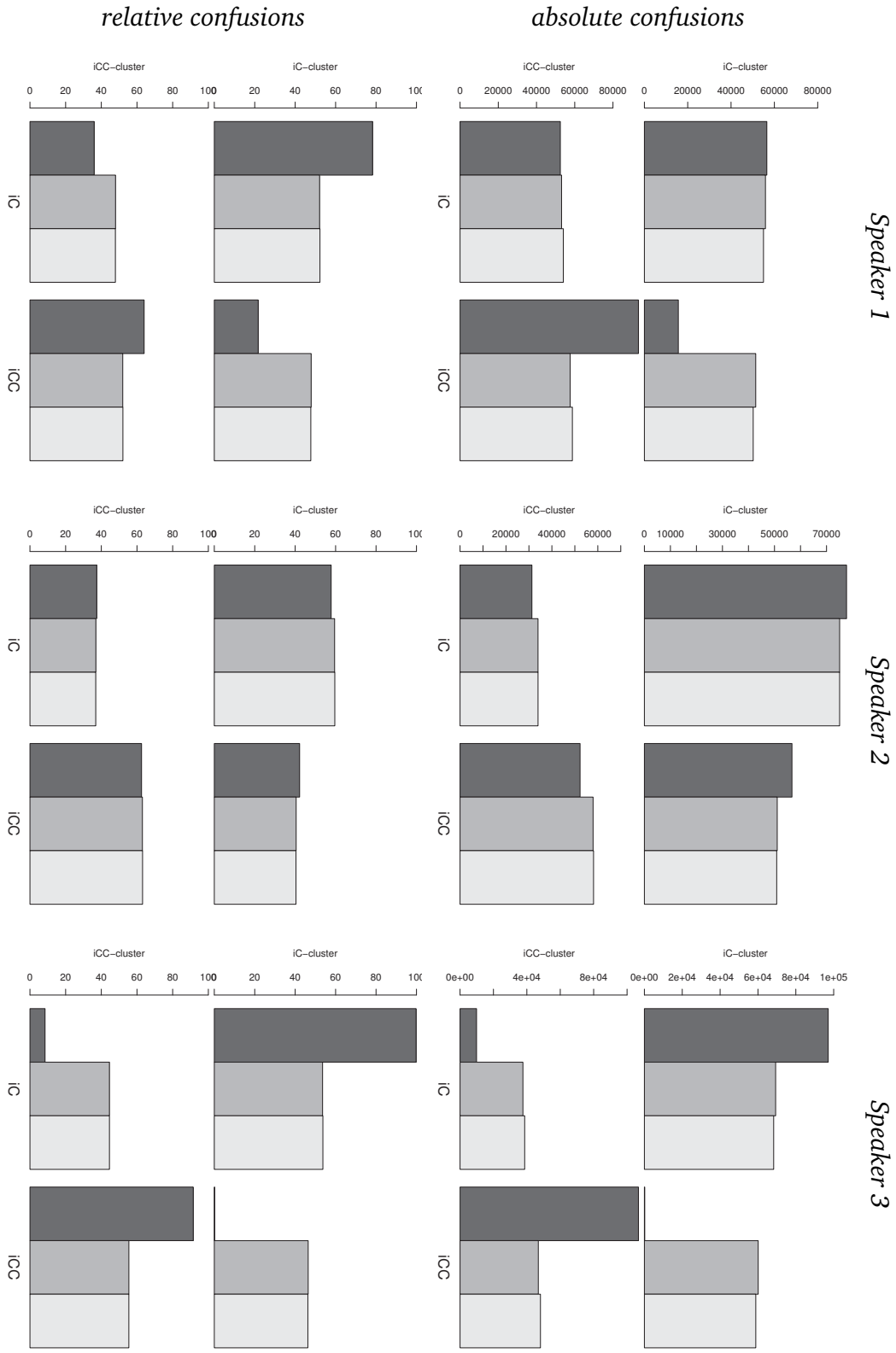


Figure 5.11.: Experiment 2.3: Confusion graphs on matched confusion tables. The top row visualises the confusions' absolute values and the bottom row shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars to acoustic data, and the medium grey bars to the combined data type.

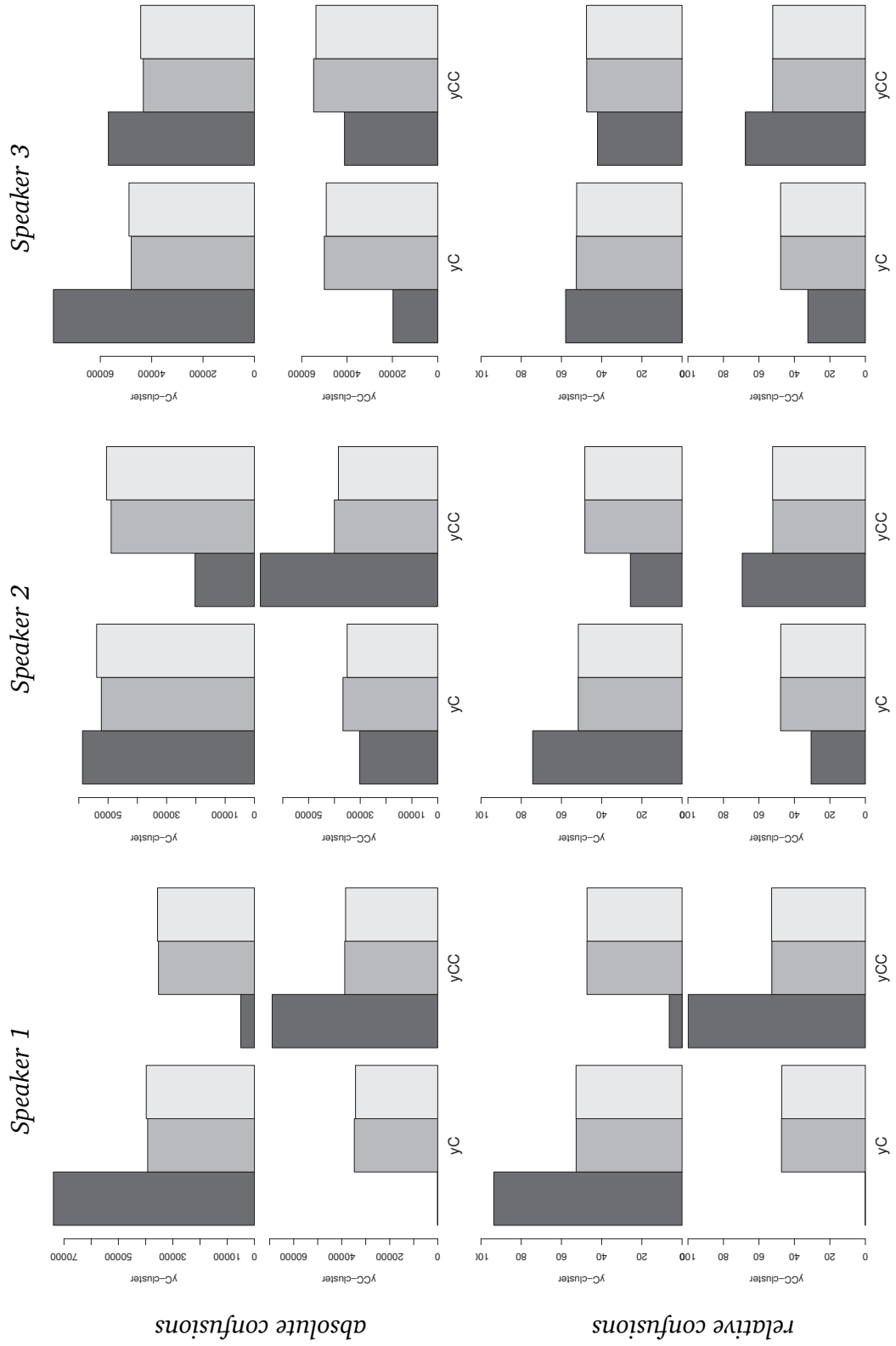


Figure 5.12.: Experiment 2.4: Confusion graphs on matched confusion tables. The top row visualises the confusions' absolute values and the bottom row shows the relative distributions for each cluster. Dark grey bars correspond to articulatory data, light grey bars to acoustic data, and the medium grey bars to the combined data type.

5.5. Conclusions and discussion

The results of experiment 1 show that continuous EMA as well as acoustic speech recordings provide useful information about phonetic segment identity in a frame-by-frame clustering scenario. For two of the three examined speakers, the articulatory data yields the best results in terms of overall clustering quality according to the measures *purity*, *V-measure* and the *adjusted Rand index*. For one of the three speakers, the combination of articulatory and acoustic data gives better results than the two uni-modal data types. A detailed analysis of the average cluster sizes by majority classes shows that the consonant clusters on acoustic data tend to be smaller than the real class size. In contrast, the vowel clusters tend to be larger than the reference class size. With articulatory data, on the other hand, no such relation between cluster size and segment type is visible. The confusion graphs which have been generated for each speaker show a tendency of less confusion (i.e. higher cluster purity) for consonants with articulatory data in comparison to acoustic and multi-modal data. The reversed picture can be observed for vowels where acoustic and combined data tend to produce less confusions. The experiment incorporated two parameters based on prior knowledge: (a) the true number of phonetic segment classes and (b) the segmentation provided by the manual annotation of the corpus. In particular, no labels were used by the clustering procedure and no specific cues were extracted from the signals. Thus, the results of experiment 1 indicate that it could be possible to model speech perception without extraction of distinct phonetic cues. Acoustic and articulatory landmarks are embedded in the spectral and continuous gestural information and they need not always be accounted for explicitly in speech modelling.

Overall, the findings of experiment 1 allow the conclusion that it can be advantageous to use unlabelled, continuous articulatory data in phonetic simulation studies. Additionally, they support models of speech perception based on a rich representation where every bit counts, such as the Context Sequence Model (CSM), discussed in section 6.2. Continuous acoustic and articulatory information can be employed in acquisition models without incorporating an a priori given set of discrete features or landmarks which would require further theoretical justification. A practical consequence is that the laborious task of full manual annotation of the data can be reduced to what is required for large scale modelling and simulation experiments and their direct, external evaluation.

The results of experiment 2 show the presence of right context information in vowel segments. Based on earlier investigations on C-centre behaviour in obstruent–sonorant clusters in Polish, one would not expect to see such coarticulatory influence of the word-final consonants on the preceding vowel. The sonorant in the investigated contexts is analysed phonologically as being excluded from the syllable and the vowel serves as a temporal anchor for the sequential organisation of the following consonant clusters. These findings seem to support models of speech perception/production with rich and context-sensitive representations, such as the CSM.

5.5.1. Problems of the present experiments and possible future work

This chapter presents clustering experiments which were performed on a corpus of acoustic speech and EMA data. The goal of these experiments was not primarily to investigate the absolute quality of a particular clustering method, but to compare the clustering results on three different types of data: (1) articulatory data as represented by EMA measurements, (2) acoustic speech recordings and (3) a combination of these two.

The results contained some very low numbers (if, for example, compared to state-of-the-art performance in ASR systems). Comparisons of the results of experiments 1 and 2 against baseline systems which operate on chance-level indicate that some structural information was captured by the clusterings that can be related to linguistic categories. Kello and Plaut (2004, p.2357) state one of the goals of their simulation experiment as determining "a lower bound on the phonetic information available in the articulatory recordings". This interpretation should also be applied to the experiments presented in this chapter. They were designed as exploratory studies and they provide ample possibilities for future work.

No detailed significance tests were computed. Additional evaluations are needed to investigate whether the clustering results reflect real differences. Indirect evaluations might be applied such that the clustering results are incorporated into another simulation. An important follow-up study to experiment 2 could be an empirical perception experiment with cross-splicing on the vowels in /ik/–/ikr/-type contexts to see whether the differences are perceptually relevant to human listeners.

The data used in the experiments presented in this chapter was taken from a corpus which has been developed for a specific purpose. It was not created for the clustering experiments. It would be interesting to the questions addressed in experiment 2 to investigate a broader range of vowels in such VC and VCC contexts. Here, there was only enough data available in the corpus for the vowels /i/ and /ɨ/. Other vowels, e.g. the two Polish mid-vowels /ɛ/ and /ɔ/ are not contained at all in the data set. The dependence on manual annotations for evaluation limited the amount of useful data for both experiments. The utterances in the Polish EMA corpus are repeated carrier phrases where only one word is varied systematically. Thus, a follow-up study to experiment 1 could be carried out investigating the full recordings of all utterances. For this purpose, the remaining data of the corpus — or a sub-set of it — would have to be segmented and labelled for evaluation.

The observed differences between the speakers deserve further attention. They could be caused by gender differences (two speakers are female, one is male), or by other individual differences like the fact that one speaker is an early bilingual while the other two are not[25].

[25]Unpublished data.

From a technical point of view there is not much difference between experiment 1 and experiment 2. They share most of the MATLAB functions. Apart from the different input data, the clustering problem is the same. This illustrates one important aspect of this thesis: In simulation experiments in phonetics and phonology, the primary focus is always on the linguistic questions. Many different simulation methods may be applied to address the same question, and, as shown here, the same method may be applied to address different questions.

The experiments presented in this chapter did not address the specific features of the acoustic speech signal or the articulatory movements which facilitate the determination of similarity between acoustic and articulatory events. The data representation was chosen such that structurally similar data was used for all parameter settings. The effective sampling rate of 250 Hz might have been one reason for the bad results with the acoustic data. However, since a frame-by-frame clustering was applied, a higher time resolution, i.e. a larger number of frames per second, would not have a huge impact on the results[26]. Still, this needs to be checked. The particular representations could also be re-considered. In particular, the combined data type was a simple "concatenation" of the two uni-modal data types. This doubled the dimensionality of the combined signals in comparison to the other data types. A weighting scheme could be applied to discriminate articulatory and acoustic information, but this would necessarily lead to the incorporation of additional linguistic knowledge into the data.

The purpose of the clustering experiment described in section 5.3 of this chapter was to examine the similarities between different phone types based on the corresponding EMA traces and acoustic data. It was not about the particular cognitive processes which allow humans to group percepts together (in this case phonetic segments) according to their mutual similarity or dissimilarity. The experiments presuppose the fact that humans are *somehow* able to recognise and group various percepts according to their mutual similarity. However, the fact that an unsupervised clustering method is used as opposed to a supervised classification method provides for a certain degree of psycho-linguistic plausibility (cf. discussion in section 3.1 on page 65).

Clustering methods are often formulated such that they tend to generate equally sized clusters. Huge differences in the sizes of the reference classes could introduce problems due to data sparsity for the learning procedure. Therefore, by applying a random sampling of the corpus data, disturbing effects attributable primarily to the specific features of the clustering algorithm are reduced. On average, any random bias of one given corpus sample set should be compensated[27].

[26] Note that a higher resolution would not have been possible for the EMA signals as they are originally stored at a sampling rate of 250 Hz in the corpus.

[27] This assumes that the used random number generator is truly random and not biased itself. Strictly speaking, this is not the case for any common implementation of a random number generator. Apart from computational efficiency, one reason for this is that it is often desirable to have an implementation which can be set to reproduce a specific sequence of pseudo-random numbers in order to be able to

The bisecting k-means algorithm used in experiments 1 and 2 operates like a top-down hierarchical clustering method. The implementation could be modified such that a corresponding tree structure is generated. Such trees can provide further insight into the properties of the investigated data. Alternatively, other hierarchical clustering methods could be applied in a follow-up study to experiment 1. The hierarchical organisation of the clusters can be an interesting object of research, itself. Linguistically, the segments investigated in experiment 1 can be grouped according to several levels of description. On top of such a hierarchical tree could be the distinction between vowels and consonants, below that categories like obstruent, sonorant etc. Comparing automatically generated hierarchical structures against such knowledge-based analyses of the available data can provide starting-points for further study.

A hierarchical analysis of the vowel data investigated in experiment 2 is probably not very useful. For both experiments, however, a variety of different clustering methods might be considered and tested in additional clustering experiments. Basic k-means, as pointed out above, cannot reliably identify certain topological structures. Other clustering methods — including both hard and fuzzy clustering — could be tested and the results of different methods could be compared.

The Polish EMA corpus, from which the speech data was taken, was designed for a very specific purpose. As this is the case for all speech databases, the theoretical and practical implications of the database structure and content must be considered. The process of category formation, which is very likely incremental, takes place primarily during first language acquisition. Simulating such a scenario would ideally require a database of child directed or child witnessed speech. With respect to second language acquisition or language change, a simulation of the formation of new categories would have to consider the existence of a set of previously established categories. If the speaker's own productions are to be integrated in a simulation, a multilingual database comprising productions of at least one speaker in more than one language would be required.

faithfully reproduce the results of a simulation experiment.

6. Articulatory data in a speech production model with a rich memory

"A category can only 'exist' by virtue of its context."

(*Hawkins, 2003, p.386*)

In this chapter I present a series of computer simulation experiments which investigate the effects of incorporation of articulatory information into the Context Sequence Model (CSM) (Wade et al., 2010). This chapter is structured as follows: First, a brief introduction to Exemplar Theory as applied in phonetics and phonology is given in section 6.1, followed by a short description of the basic CSM in section 6.2. The next sections present my experiments with two different speech databases that consist of both digital speech recordings and articulatory measurements obtained by EMA. Section 6.3 presents experiments with the *Polish EMA corpus* and section 6.4 presents experiments with the English *MOCHA* corpus. General theoretical as well as methodological implications are discussed in section 6.5 based on the presented simulations.

Some results of the experiments discussed in this chapter have been presented previously (Duran et al., 2011b,c, 2012a). Note that no MFCC data was used and only the "emph" part of the Polish corpus was examined in those studies. The results presented here are consistent with the previously reported ones. They are, however, based on a new, expanded and more elaborated implementation of the experiment.

6.1. Exemplar Theory

Exemplar Theory was first introduced in linguistics in the fields of phonetics and phonology for the modelling of categorisation in speech perception. An early exemplar-theoretic model has been proposed by Lacerda (1995). In a meta-study, he re-examined data from empirical studies of the perceptual magnet effect (cf. section 2.4.1). His computational model is based on two assumptions: first, that perceived speech exemplars are stored

individually in memory, and second, that a labelling function is available that associates exemplars with category labels (p.142). In a "simple numerical exercise", Lacerda shows that the perceptual magnet effect can be modelled without recourse to explicitly defined prototypes. Another exemplar-theoretic approach to the perceptual magnet effect with computer simulation experiments is presented by Shi et al. (2010, p.449). They propose an implementation of the Bayesian model developed by Feldman et al. (2009a) which does not require category labels and report on a series of simulation experiments. They conclude:

> "Our results suggest that exemplar models are not simply process models, but rational process models — an effective and psychologically plausible scheme for approximating statistical inference."
> (Shi et al., 2010, p.462)

The model developed by Lacerda (1995) exhibits the essential properties of exemplar-theoretic models: categories are represented as collections of previously perceived tokens stored in a detail-rich memory. All tokens, or *exemplars*, retain their fine auditory/phonetic details. Additionally, they may be associated with category labels. Recognition is based on similarity relations, i.e. on comparisons of new percepts against the remembered exemplars. The memory is constantly updated, adding new percepts to the exemplar collections. Pierrehumbert (2001, p.139) speaks of exemplar *clouds*. She assumes that exemplars are organised in a cognitive map such that instances of one category are located close to each other.

Johnson (1997a) developed an exemplar-theoretic model of speech perception which does not rely on speaker normalisation. He proposes that exemplars are associated with sets of labels, including extra-linguistic information like speaker identity (p.147). Important parameters of the model include: an attention weight for some 'auditory property', a base activation for some exemplar, a sensitivity constant and noise added to activation levels. He describes simulation experiments with a set of vowel tokens. Each of the tokens is in turn taken from the memory and treated as an unknown input percept. A vowel confusion matrix based on the model's categorisations is compared against empirical results. In summary, Johnson's (1997a, p.162) model shows speaker normalisation behaviour. His exemplar-theoretic model contrasts with common normalisation approaches which are based on changing the representations. Speaker normalisation is an emergent property of the model employing complex exemplar representations. Speaker-specific variability is thus not treated as noise but incorporated into the perception model.

Pierrehumbert (2001) presents a model of speech perception and production in which lexicon and grammar are "two degrees of generalization over the same memories" (p.139). Pierrehumbert (2001, p.142) emphasises that exemplar-theoretic models can account for systematic and probabilistic variations in fine phonetic details, depending not only on language and dialect but also on factors such as word frequency. Postulated universal symbolic features, she claims, cannot achieve this. Unlike the two models presented above, Pierrehumbert (2001, p.143) presents simulation experiments on a

model of speech production. She assumes that production is based on the selection of an exemplar from memory. Two important principles of her model are examined in simulation experiments: lenition and entrenchment. Lenition is implemented as a systematic bias in production which slightly shifts each production from the initially selected target exemplar (p.9). Entrenchment is implemented in exemplar selection such that not only one selected exemplar represents the production target. Instead, all exemplars in a defined neighbourhood around a selected target location within an exemplar cloud contribute to the properties of the production (p.11).

Hawkins (2003) examines the relation between fine phonetic detail and the meaning associated with an utterance. She argues that speech units cannot be separated from their context, since it is the context and its relation to the phonetic details of the speech signal which defines "whether the whole percept is coherent or not, and hence what each 'unit' is" (p.386). Exemplars, Hawkins (2003, p.387) suggests, can be stored in more than one way and associated with different kinds of information. Particularly, she points out that speech exemplars are not necessarily represented and/or processed as individual segments:

> "There is no necessary segmentation of signal into formal linguistic categories: segmentation and categorization emerge as a result of the way the brain naturally organizes and classifies like with like. Because categories are self-organizing and emergent, each individual develops somewhat different mental representations of language."
> (Hawkins, 2003, p.389)

An exemplar-theoretic model of category competition is presented by Wedel (2004a). He argues that *contrast* is not "a property of forms" but that it is rather implicitly driven by the statistical association of forms to categories (p.1). He puts forward the hypothesis that "[...] contrast maintenance is driven through category competition between form-meaning pairings[...]" (p.4) and investigates this hypothesis in a series of simulation experiments.

Johnson (1997a) addresses the *head-filling-up problem*[1] — the seeming impossibility of an ideal exemplar model which would require storage of *all* perceived items. The problem is partially mitigated by reference to the observed human ability of remembering large numbers of pictures. Johnson (1997a, p.150) assumes additionally that the perceptual space may be quantized based on just noticeable differences, i.e. reflecting the fact that humans cannot perceive arbitrarily small differences along auditory dimensions. Pierrehumbert (2001, p.4) assumes that memories decay over time and also that the representations are quantized. The simulations presented by Wedel (2004a, p.3) avoid this problem by deleting one exemplar from the memory for each new addition.

[1] Johnson (1997a, p.146) attributes this term to Neal Johnson.

6.2. The Context Sequence Model

The experiments presented in this chapter are based on the Context Sequence Model (CSM). This is an exemplar-theoretic model developed by Wade et al. (2010). It extends earlier work by Wade and Möbius (2007)[2]. The CSM seeks to describe how speech production is guided by implicit phonetic knowledge stored in a context preserving episodic memory of remembered speech items.

Wade and Möbius (2007, p.402) present an exemplar-theoretic model of speech perception which accounts for speaking rate effects without explicitly tracking speaking rate or applying any rate normalisations. The model operates on acoustic cues extracted at specific landmark positions from a detail-rich memory of speech exemplars. Newly perceived speech items are identified by a context-sensitive comparison of these landmarks against the landmarks of exemplars stored in memory. Their simulation demonstrates how speech perception can be modelled in exemplar-theoretic terms without the need for substantial abstraction like segmentation or structural analysis or explicit normalisation processes (p.402). A central aspect of the model is the incorporation of surrounding context information into the exemplar comparison procedure.

The main property of the CSM is the assumption of a detail-rich, context-preserving sequential memory. Such a memory structure was used in the simulations by Wade and Möbius (2007), and described as follows:

> "[...] perception makes use of a memory containing an ordered collection of richly specified, real-time acoustic descriptions of previously perceived sounds, not unlike a continuous recording of previous auditory input."
> (Wade and Möbius, 2007, p.402)

The memory is proposed to contain sequential information which retains original co-ocurrences of perceived speech items. An essentially continuous, non-hierarchical structure is envisioned by the authors which could contain multi-modal information. Their simulation experiments, however, employ "traditionally described feature or segment labels" (p.403) for the sake of simplicity. Wade et al. (2010) describe the basic features of such a memory as follows:

> "In a series of simulations [...], we proposed that exemplars consist of sequences that represent stretches of speech much longer than the units of interest, perhaps corresponding to entire utterances. These sequences preserve both local, detailed spectral and amplitude information and the temporal organization of this information."
> (Wade et al., 2010, p.228)

What distinguishes the CSM from other exemplar-theoretic models of speech production and perception is that the exemplars are not assumed to be instances of speech

[2] Note that the model presented by Wade and Möbius (2007, p.402f) shares many important aspects with the model implemented in later simulation experiments, although the term "*Context Sequence model*" is not used in publications by the authors prior to Wade et al.'s (2010, p.227) paper.

items at one particular level of linguistic abstraction (like phones, syllables or words) which are stored in isolation. Instead, the CSM postulates a rich, continuous memory where previously perceived speech items are stored embedded in their original context.

The experiments presented in this chapter are especially based on Wade et al.'s (2010, p.233f) experiment 2. Speech production is modelled in the CSM as a process of sequential selection of speech items from the memory of previously processed speech exemplars. Selection of one specific exemplar for production is based on token weighting according to the similarity of the current production context, and the context of the remembered speech items in memory. Context on both sides of an exemplar has to be taken into account. The left context consists of the acoustic details of the items produced immediately before the current production target. The right context involves an estimation of what is going to be produced immediately after the current target. For the sake of simplicity, Wade et al. (2010, p.229f) implemented the simulation such that the right context is represented by a "linguistic" context encoding the categorical information of the following segments (i.e. their label).

The context match function is defined as (Wade et al., 2010, p.217):

$$\text{c-match}(t_0, t_e, n_a, n_l, n_e) = \exp\left\{\sum_{d=1}^{D_A} A_{d,t_e-n_a:t_e-1} \cdot A_{d,t_0-n_a:t_0-1} + \sum_{d=1}^{D_L} L_{d,t_e+n_e:t_e+n_e+n_l} \cdot L_{d,t_0+n_e:t_0+n_e+n_l}\right\} \quad (6.1)$$

where $A_{d,m:n}$ is the dth dimension of the acoustic memory sequence from index m to n. Analogously for the linguistic memory L. The indices t_0 and t_e refer to the beginning of the exemplar under consideration and the exemplar currently being produced, respectively. n_a is the length of the left acoustic context, n_l is the length of the right linguistic context and n_e is the length of the target segment. The dimensionality of the acoustic and the linguistic memory is specified by D_A and D_L.

Originally, the CSM was designed to incorporate only auditory information in its memory representation. My experiments presented in sections 6.3 and 6.4 enrich the memory with articulatory information by inclusion of continuous EMA measurements. Wade et al. (2010, p.238) argue for the importance to address articulatory processes in order to develop a more complete account of phonetic competence. They suggest that articulatory data should be used in addition to acoustic data. This is the starting point of the simulation experiments in the following sections.

6.3. Experiment on Polish data

In the first simulation experiment, the role of articulatory information is investigated when integrated into the process of exemplar selection of an implementation of the CSM. For this purpose, digital speech recordings and articulatory data represented by EMA measurements of Polish speech is integrated into the model.

This experiment is motivated by considerations from exemplar theory where all feedback during speech production, including articulatory habits, is assumed to be memorised in detail in a perception–production feedback loop (cf. section 2.4 on page 51). These stored percepts are assumed to provide a basis for future productions of speech items.

The main aim of the simulation experiments presented in this section is to investigate the application of the CSM to articulatory data and to compare the model's performance on different types of data. For this purpose, the same simulation experiment is carried out on three data types: First on acoustic data taken from digital speech recordings. This can be considered the baseline since it corresponds to the earlier implementation of the CSM model by Wade et al. (2010, p.232f). Second, the simulation is carried out on articulatory data, taken from EMA measurements. And finally, the simulation is carried out on a combined data type incorporating both acoustic and articulatory data.

The remainder of this section is structured as follows: Section 6.3.1 provides information about the speech material used in the simulation experiments. The implementation is described in section 6.3.2. A discussion of the evaluation and the results concludes this section.

6.3.1. Speech material

The speech material for this experiment is taken from the Polish EMA corpus, which is described in detail in appendix A.1 on page 251. The corpus consists of a collection of target words which are embedded into carrier phrases. Only consonants or consonant clusters in onset or coda position of the target words were of interest in the original study (Bruni, 2011, p.111ff). Therefore, only the consonants and the vowel of the respective syllables in these target words are labelled at the phonetic level, and, only these labelled phone segments are used in the study presented in this section (along with stretches of 0.5 s of the signal(s) preceding the first segment to provide a left context for it). The labelled segments of both parts of the corpus are taken as the segmental production targets. This amounts to $281 + 281 + 280$ segments in total for the three speakers in the "emph" part of the corpus, and to $283 + 280 + 280$ segments in total for the three speakers in the "noemph" part of the corpus.

The acoustic data is encoded by amplitude envelopes (as in the clustering experiments, cf. section 5.3). A more commonly used representation of speech signals (especially in the field of automatic speech processing) are MFCCs. Therefore, the simulation

has also been applied to the corpus where the acoustic data is encoded by an MFCC-representation (as in the segmentation experiments, cf. section 4.4.4).

Pre-processing

The corpus data was pre-processed and stored in binary format to allow fast access for the simulation scripts. The acoustic speech signal was re-sampled to 250 Hz for computational efficiency and, more important, in order to obtain similar data representations for both acoustic and articulatory data. Following Wade et al. (2010, p.232), the acoustic signal was converted to an 8-dimensional representation of amplitude envelopes taken from 8 logarithmically spaced frequency bands which were delimited by the following cut-off frequencies: $[80, 142, 253, 450, 801, 1425, 2535, 4509, 7999]$. Additionally, 13-dimensional MFCCs were computed using the `mfcc` function of the Auditory Toolbox (Slaney) for MATLAB. The articulatory data is taken from horizontal and vertical movements of sensors placed and the tongue body 1 cm, 3 cm and 4 cm behind the tongue tip, and from sensors placed on the vermillion border of the upper and lower lip (one signal corresponding to lip aperture, cf. Bruni, 2011, p.115). The data is stored originally at a sampling rate of 250 Hz in the corpus, so no re-sampling is applied to the EMA signals. The corresponding velocity and acceleration data for both acoustic and articulatory signals is computed using MATLAB's `diff` function. Each data frame is thus represented by a 24-dimensional vector[3] which is then length-normalised.

6.3.2. Implementation

The simulation experiment uses the continuous acoustic and articulatory signals analogously to the original implementation of the CSM as developed by Wade et al. (2010, p.232f). Thus, the representation does not use features extracted at specific "landmark" positions of the signal (like, for example, in the simulation experiments by Wade and Möbius, 2007), but the entire digitised continuous signal. Production of an utterance from the corpus is simulated in this experiment by taking that given utterance out of the corpus and using it as a production target. The remaining data in the corpus is treated as the memory. This process is repeated for all utterances in the corpus. A similar approach is used in the experiment by Johnson (1997a, p.155), where a corpus of English vowels is treated as an exemplar memory and one vowel at a time is taken aside to serve as an unknown token for an identification simulation.

The simulation experiment is implemented and carried out in MATLAB, version 7.8.0 (R2009a), 64-bit, on Linux. The source code for this experiment is shown in appendix B.3.1, starting on page 383. An outline of the simulation experiment is shown

[3] The 24 dimensions are composed as follows: eight amplitude envelopes plus their first and second derivatives for acoustic data; and three measurements of tongue movements and one for lip aperture plus the respective first and second derivatives for (horizontal and vertical) articulatory data.

in pseudo-code in algorithm 6.1. As shown in this algorithm, the experiment consists essentially of three nested loops[4]. The input is a phonetically annotated sub-set of the corpus which contains data from one single speaker. Thus, the sub-corpora for the three speakers in the Polish EMA corpus are separated and the simulation experiment considers only one speaker at a time. Cross-speaker effects are not considered in this experiment[5].

The first loop iterates over all data types. The symbol "AC" in algorithm 6.1 denotes acoustic data, "AR" denotes articulatory data and "ACAR" the combined representation. The second loop iterates over all utterances of the corpus, i.e. the carrier phrases into which the target words have been embedded. The production targets for this simulation experiment are the labelled phone segments from the Polish EMA corpus. In order to avoid selection of segments from the original utterances, each utterance in the corpus is in turn defined as a production target and excluded from the model's memory. This forces the model to select a segment from a different utterance. Note, that this also means that there will never be a perfect match as there are no identical acoustic/articulatory contexts in the memory at the signal level. Finally, the third loop iterates over all segments from the current production target sequence and simulates the sequential production based on context matching and candidate selection from the memory sequence.

The main MATLAB script is `runCSMEMA` (see source code listing B.60 on page 383). This script initialises the simulation parameters for the "emph" part of the corpus and it specifies the input and output files and directories. The script `runCSMEMAnoemph` sets the parameters for the experiment on the "noemph" part of the corpus (listing B.61 on page 385). The actual experiment is implemented in the MATLAB function `csmProduction` (listing B.62 on page 387). This runs the experiment for all parameter settings, i.e. for all speakers and data types, and evaluates the results.

The function `createMemorySequence` (listing B.64 on page 393) extracts all segment meta data and all signal data required for the experiment from pre-processed corpus files. It generates a matrix `corpus` which represents the model's *memory sequence*. The memory sequence is represented as a $(N \times D)$-matrix, where N is the number of frames and D is the number of dimensions (e.g. 24 in the case of 8 amplitude envelopes plus their corresponding first and second derivatives). The function also generates two matrices containing the annotations. The $(N \times 6)$-matrix `labels` contains

[4] Note that only the essential aspects of the implementation are presented and discussed in this section. The source code listings in appendix B all contain detailed in-line comments. The reader is referred to these source code listings for full implementation details.

[5] The hypothesised memory will in general contain exemplars originating from perceived speech items uttered by various speakers. In exemplar-theoretic models, these exemplars can also serve as production targets. The experiment presented in this section does not consider such phenomena and uses only memorised exemplars from the same speaker (or *ego exemplars*, as Johnson, 1997a, p.154 calls them) as potential candidates for production .

Algorithm 6.1 CSM production experiment

```
1:  Require: Subset of Polish EMA corpus for one speaker
2:  for all data types ∈ {AC, AR, ACAR} do
3:      for all utterances U in corpus do
4:          Let production target T = [t_1 ... t_n] be the sequence of phonetic segments
            of labelled target word from U
5:          Let the memory M be the entire corpus excluding U
6:          Initialise output sequence O with 0.5 s from the left context of the original
            signal preceding segment t_1
7:          for all t_i ∈ T do
8:              Let the left context of t_i be the last 0.5 s from O
9:              Match left context of t_i with left contexts of all candidate exemplars
                from memory
10:             Select highest scoring candidate for production and copy it from M to
                output sequence O
11:         end for
12:     end for
13: end for
```

word, phone and file labels for each frame, and the $(S \times 5)$-matrix `segments` contains the segments' start and end indices, their associated labels and context information, where S is the total number of segments in the corpus[6]. The present implementation is independent of the specific structure of the underlying corpus. The data structures created by the function `createMemorySequence` can also be extracted from any other corpus that contains a digital signal and associated labels at the segment level. Some suitable definition of utterances is also assumed from which the production targets can be derived. Section 6.4 (page 234) presents results of this experiment applying basically the same implementation to a fairly different corpus.

The inner-most loop `for t=1:size(segments,1)` in function `csmProduction` (B.62) iterates over all production targets. For the actual implementation, the two innermost loops from algorithm 6.1 are merged into one. The current utterance is not specified explicitly. The candidate cloud generated for each target segment is taken from the memory sequence ignoring the one utterance associated with the current target (making use of MATLAB's efficient matrix indexing and processing capabilities). A phone label index "-1" in the list of production targets indicates a left-context segment. These are actually not produced at the very beginning of a target utterance but copied directly

[6] The data structures generated by the function `createMemorySequence` (as wells as some other data structures in this simulation) show a high degree of redundancy. As the main purpose of the studies presented here is not a memory efficient implementation, such redundant structures are used to improve time efficiency of the simulation as well as readability of the code.

to the output sequence. A phone label index "0" indicates a right-context segment, i.e. the end of the current utterance. In the present implementation, these right-context segments are not further processed. Other values correspond to the phone label string and represent actual production targets which need to be produced by the model.

Each utterance in the corpus is taken in turn as a production target from which the sequence of phone segments $T_U = [t_1, \dots t_n]$ is taken from the phonetically labelled segments (with $n \leq 3$). The output sequence is initialised by copying 0.5 s of the acoustic and/or the articulatory signals which immediately precede the first segment t_1. This is interpreted as the original left context of the first segment that is to be produced by the model. Then, for each target segment $t_i \in T_U$, a stretch of 0.5 s from the output sequence provides the left context for the current segment.

The function `cmatch` (listing B.66 on page 407) implements equation 6.1 and computes the similarity between the context of the candidate and the context of the current reference segment. This is computed for all exemplars in the cloud of candidates. The one exemplar with the highest c-match score wins and is selected for production. This function incorporates an additional major modification to the original CSM, namely, that this simulation does not use the right, i. e. the "linguistic" context to match the context of the target segment with the candidate exemplars' contexts. This is done due to the relatively small size of the corpus and its regular and, therefore, highly predictable structure. In order to avoid an unwanted selection bias, the right context is not considered. Exemplar selection in this scenario is more difficult as it has to rely solely on the raw acoustic and/or articulatory signal information of the left context.

The combined data type is a simple concatenation of the dimensions of the two individual data types. This is equivalent to summing the c-match scores for the two separate data types. Thus, articulatory and acoustic data are weighted equally in case of the combined data type.

Despite the fact that this experiment is based on the exemplar-theoretic assumption that all feedback during speech production is stored in memory and immediately available for future productions, the produced utterances in the simulation are not added to the corpus. For the sake of simplicity and in order to avoid artefacts, the underlying memory representation is not changed. The simulation has thus to be interpreted as a static simulation for each produced utterance which does not take into account such processes as memory decay or interference effects or any other kind of individual language change over time. The corpus is treated as a snapshot of the memory at one given moment in time. As a consequence the actual order of the utterances and their production is not relevant to the results of the simulation.

left context accuracy ca^{l}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.386	0.893	0.904	0.915	0.897	0.925
Speaker 2	0.385	0.918	0.904	0.961	0.918	0.925
Speaker 3	0.387	0.939	0.954	0.964	0.943	0.961

right context accuracy ca^{r}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.558	0.797	0.769	0.847	0.801	0.808
Speaker 2	0.553	0.829	0.822	0.875	0.829	0.829
Speaker 3	0.553	0.829	0.850	0.907	0.829	0.850

left-and-right context accuracy ca^{lr}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.222	0.705	0.698	0.772	0.712	0.747
Speaker 2	0.221	0.754	0.744	0.840	0.754	0.765
Speaker 3	0.220	0.782	0.811	0.875	0.786	0.814

Table 6.1.: Context accuracy on Polish data from the "emph" part of the corpus for both audio representations using amplitude envelopes ENV and MFCCs and the respective combined data types with the EMA data.

6.3.3. Evaluation and Results

The results were evaluated on the segment as well as on the syllable level, using the manually created annotations of the target segments as the reference.

Segment-level evaluation

The *context accuracy* is computed for the segment-level evaluation. It is defined as the proportion of produced segments for which their original context matches the production context. If, for example, a selected [p] segment was taken from of a stored [...upr...]

left context accuracy ca^{l}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.388	0.922	0.922	0.919	0.922	0.926
Speaker 2	0.387	0.968	0.946	0.979	0.968	0.946
Speaker 3	0.387	0.907	0.918	0.968	0.907	0.929

right context accuracy ca^{r}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.557	0.841	0.795	0.855	0.841	0.807
Speaker 2	0.553	0.861	0.825	0.879	0.861	0.832
Speaker 3	0.553	0.846	0.825	0.900	0.846	0.836

left-and-right context accuracy ca^{lr}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.223	0.777	0.742	0.795	0.777	0.760
Speaker 2	0.220	0.832	0.775	0.857	0.832	0.782
Speaker 3	0.220	0.768	0.757	0.871	0.768	0.779

Table 6.2.: Context accuracy on Polish data from the “noemph” part of the corpus for both audio representations using amplitude envelopes ENV and MFCCs and the respective combined data types with the EMA data.

context to be produced in a [...ɨpr...] context, the right production context matches the [p] segment's original right context but the left context does not match. The context accuracy is computed for matching only left contexts (ca^{l}), matching only right contexts (ca^{r}) and matching both left and right contexts (ca^{lr}).

More precisely, the context accuracy is defined as follows. Let C_l be the set of all produced segments for which the label of their preceding segment in the production sequence T is identical to the label of the preceding segment of the corresponding original exemplar in the memory sequence $\mathcal{M}$. Let analogously C_r be the set of all produced segments with a matching following context in both T and $\mathcal{M}$, and let C be the set of all produced segments for which both the preceding and following contexts match. More formally, I define:

$$C_l = \{t_i : L(t_{i-1}) = L(\phi(t_{i-1})) \quad \wedge \quad t_i \in T \quad \wedge \quad 1 \leq i \leq |T|\} \tag{6.2}$$

$$C_r = \{t_i : L(t_{i+1}) = L(\phi(t_{i+1})) \quad \wedge \quad t_i \in T \quad \wedge \quad 1 \leq i \leq |T|\} \tag{6.3}$$

$$C = C_l \cap C_r \tag{6.4}$$

where $\phi(t)$ is a function that maps a produced segment $t \in T$ to its original exemplar in memory $t' \in \mathcal{M}$, $L(t)$ maps a segment t to its class label, and $|T|$ is the total number of segments in the production sequence T.

Based on the sets defined above, the three context accuracy measures can be defined as:

$$ca^{\mathrm{l}} = \frac{|C_l|}{|T|} \tag{6.5}$$

$$ca^{\mathrm{r}} = \frac{|C_r|}{|T|} \tag{6.6}$$

$$ca^{\mathrm{lr}} = \frac{|C|}{|T|} \tag{6.7}$$

Equations 6.5, 6.6 and 6.7 give the proportion of produced segments $t \in T$ for which their corresponding original segment $t \in \mathcal{M}$ has an identical context, in terms of the associated segment label, out of the total number of produced segments. Thus, the value can be interpreted as a usual accuracy measure, ranging from 0 to 1. In order to be able to interpret the accuracy values, it is necessary to compare them against a baseline. In this experiment the baseline is defined as a random selection of a segment from the set of candidates (i.e. from the set of segments with the same label). The baseline context

accuracy values are determined as follows:

$$base\text{-}ca^{\mathrm{l}} = \frac{\sum_{i=1}^{|T|} n_i^l}{\sum n_i} \tag{6.8}$$

$$base\text{-}ca^{\mathrm{r}} = \frac{\sum_{i=1}^{|T|} n_i^r}{\sum n_i} \tag{6.9}$$

$$base\text{-}ca^{\mathrm{lr}} = \frac{\sum_{i=1}^{|T|} n_i^{\mathrm{lr}}}{\sum n_i} \quad , \tag{6.10}$$

where n_i^l, n_i^r and n_i^{lr} are the numbers of available candidates in memory for target item i with a correct matching left context, a correct matching right context and a correct matching left and right context, respectively. The total number of available candidates in memory for a target item i is n_i. And again, $|T|$ is the total number of target segments in the production sequence T. The baseline is computed for each speaker sub-corpus separately, but the numbers are very similar due to the nearly identical structures of the corpora. The baseline for ca^{l} is around 0.39, the baseline for ca^{r} is around 0.55 and the baseline for ca^{lr} is around 0.22. These numbers confirm that it is indeed easier to predict the correct right context for each target segment. Thus, omitting the right context from the candidate scoring procedure appears justified in this present experiment where the primary focus is on the effect of the different signal data types.

The results for the experiment on the "emph" part of the Polish EMA corpus with the amplitude envelope representation are shown in table 6.1 on page 227. The table shows that the context accuracy is consistently higher for articulatory data (column "EMA") than it is for acoustic data alone (column "ENV") or the combined representation (column "ENV+EMA"). For all three data types, the performance is clearly above the baseline. Given the fact that the right context was explicitly excluded from the context matching procedure, it is interesting to note the high values for the right-context accuracy ca^{r}. It is in absolute terms consistently lower in all cases than the corresponding left-context accuracy values. However, it is also well above the baseline for all speakers and all data types.

The results for the experiment on the "noemph" part of the Polish EMA corpus are shown in table 6.2 on page 228. As for the "emph" data, the accuracy on articulatory data (column "EMA") is in most cases higher than it is on the acoustic data with the amplitude envelope representation (column "ENV"). However, ca^{l} is slightly lower for speaker 1 on articulatory data in comparison to the acoustic amplitude envelope data. The combined signal ("ENV+EMA"), in general, shows no improvements over the acoustic data ("ENV") alone.

The results for the experiment where instead of the amplitude envelope representation for the acoustic data an MFCC representation was used are also shown in tables 6.1 and 6.2, on pages 227 and 228, respectively. The results are identical for the articulatory data in both experiments using either the amplitude envelopes or the MFCC representation. Thus, in order to avoid unnecessary repetitions the results are presented in one combined table for both audio representations. The columns "MFCC" show the results for the acoustic data and the columns "MFCC+EMA" show the results for the combined articulatory and acoustic data, using the MFCC representation. The results are similar to those based on the envelopes. Table 6.1 shows the same patterns for MFCCs on the "emph" part of the corpus as the corresponding results for the amplitude envelopes. The context accuracy is consistently higher for articulatory data ("EMA") than it is for the other data types. In some cases (but not all), the accuracy for the combined data ("MFCC+EMA") shows some improvement over the accuracy for the acoustic data ("MFCC"). Table 6.2 shows similar patterns for MFCC on the "noemph" part of the corpus as the corresponding results for the amplitude envelopes. The highest context accuracy is achieved in most cases for the articulatory data. Again, there is one exception for speaker 1. In contrast to the results based on amplitude envelopes, the combined data ("MFCC+EMA") shows some improvements over the acoustic data ("MFCC") alone.

Syllable-level evaluation

The context accuracies represent a segment-level evaluation. The performance of the simulation can additionally be evaluated at the syllable-level. Two measures are applied here, evaluating the produced output according to the syllable type and the syllable position. The syllables of the corpus can be divided into two types: there are syllables with a simple consonant onset or coda (e. g. /k/ in ⟨kadisz⟩ or /p/ in ⟨typ⟩), and there are syllables with complex consonant clusters (e. g. /kl/ in ⟨klawisz⟩ or /pr/ in ⟨Cypr⟩). Based on this classification of the target words' syllables, it can be evaluated how often the model selects a segment from the correct syllable type for production — "correct" meaning that the syllable type of the target segment in the output sequence and the syllable type of the selected segment's original syllable in the memory sequence are the same. Table 6.4 on page 233 shows individual confusion tables for the three speakers of the Polish EMA corpus. The three data types are the same as above: acoustic data, articulatory data and a combined data type. As for the segment-level evaluation, the results for the experiments using amplitude envelopes (which are again denoted by the label "ENV") and the results for the experiments using an MFCC-representation of the audio signal are presented in one combined table. The label "C" indicates syllables with a single consonant and the label "CC" indicates syllables with a consonant cluster, in either onset or coda position. Table entries on the diagonal of each individual table are correct productions. As the tables show, there is more confusion with simple C syllables,

Speaker 1			Speaker 2			Speaker 3		
acoustic (ENV)			acoustic (ENV)			acoustic (ENV)		
	onset	coda		onset	coda		onset	coda
onset	86	0	onset	84	0	onset	84	0
coda	1	82	coda	0	85	coda	0	84
acoustic (MFCC)			acoustic (MFCC)			acoustic (MFCC)		
	onset	coda		onset	coda		onset	coda
onset	86	0	onset	84	0	onset	84	0
coda	0	83	coda	3	82	coda	1	83
articulatory			articulatory			articulatory		
	onset	coda		onset	coda		onset	coda
onset	84	2	onset	84	0	onset	84	0
coda	0	83	coda	0	85	coda	0	84
combined (ENV+EMA)			combined (ENV+EMA)			combined (ENV+EMA)		
	onset	coda		onset	coda		onset	coda
onset	86	0	onset	84	0	onset	84	0
coda	1	82	coda	0	85	coda	0	84
combined (MFCC+EMA)			combined (MFCC+EMA)			combined (MFCC+EMA)		
	onset	coda		onset	coda		onset	coda
onset	86	0	onset	84	0	onset	84	0
coda	0	83	coda	2	83	coda	0	84

Table 6.3.: Syllable-position confusion tables for all 3 speakers of the Polish EMA corpus and all 3 data types: acoustic (amplitude envelopes ENV and MFCC), articulatory (EMA) and combined data. Rows correspond to target syllable positions of segments and columns to the original syllable positions of the produced segments in memory.

Speaker 1			Speaker 2			Speaker 3		
acoustic (ENV)			acoustic (ENV)			acoustic (ENV)		
	C	CC		C	CC		C	CC
C	39	16	C	44	11	C	44	12
CC	6	108	CC	11	103	CC	11	101
acoustic (MFCC)			acoustic (MFCC)			acoustic (MFCC)		
	C	CC		C	CC		C	CC
C	37	18	C	43	12	C	46	10
CC	10	104	CC	16	98	CC	8	104
articulatory			articulatory			articulatory		
	C	CC		C	CC		C	CC
C	45	10	C	43	12	C	50	6
CC	3	111	CC	8	106	CC	4	108
combined (ENV+EMA)			combined (ENV+EMA)			combined (ENV+EMA)		
	C	CC		C	CC		C	CC
C	39	16	C	44	11	C	44	12
CC	4	110	CC	11	103	CC	11	101
combined (MFCC+EMA)			combined (MFCC+EMA)			combined (MFCC+EMA)		
	C	CC		C	CC		C	CC
C	39	16	C	44	11	C	47	9
CC	4	110	CC	15	99	CC	8	104

Table 6.4.: Syllable-type confusion tables for all 3 speakers of the Polish EMA corpus and all 3 data types: acoustic (amplitude envelopes ENV and MFCC), articulatory (EMA) and combined data. Rows correspond to the target segments' syllable types and columns to the original syllable types of produces segments, where the symbol C indicates a syllable with a single consonant and CC indicates a syllable with a consonant cluster.

for which the model relatively often selects segments from original CC syllables. With complex CC syllables on the other hand, there is only little confusion visible. And the general observation is that in most of the cases the segments are selected from the correct type of syllable for production.

The produced segments can also be evaluated according to how often the model selected a segment for a given position within the syllable (either onset or coda) from a matching position in the memory. Table 6.3 on page 232 shows individual confusion tables for the three speakers of the Polish EMA corpus and the three data types. There is almost no confusion between onset and coda position for the produced segments.

In both cases of the syllable position and the syllable type confusion, there is no influence of data type visible. It seems that there is no difference between articulatory and acoustic data at this level of evaluation.

6.4. Experiment on English data

The second experiment uses the MOCHA database (Wrench, 1999). The simulation experiment is essentially the same as the one with the Polish EMA corpus presented in the previous section. The descriptions of the experiment, its implementation, evaluation and results are therefore limited to those aspects which differ from the previous section.

6.4.1. Speech material: The MOCHA-TIMIT database

The experiments presented in this section use English recordings from the MOCHA database. Details about the corpus are given in appendix A.4 on page 257. Data from one female and one male speaker is used for the present experiments.

The original audio and EMA data of the corpus was re-sampled for the first experiment to a sampling rate of 250 Hz using the `resample` function provided by MATLAB's Signal processing toolbox (The MathWorks, Inc., 2009). This was done to carry out the same simulations with a comparable input on the MOCHA database as it was done earlier on the Polish EMA corpus. Additionally, the experiments are run on data encoded at 500 Hz. This is the original sampling rate of the EMA recordings in the MOCHA corpus. This allows testing the influence of the frame rate on the experimental results. Analogously to the experiments on the Polish corpus, the acoustic data of the MOCHA corpus was converted to an amplitude envelope representation, and, additionally to an MFCC representation and the experiments are carried out twice, using bot types of audio data.

In contrast to the Polish EMA corpus, the MOCHA corpus constitutes a phonetically balanced database covering the entire segment inventory of its target language. It contains also segments labelled as "silence". These are not treated as target segments for production. Such silence segments were in some cases shorter than the specified

length of the left context, i.e. 0.5 s. As a simple solution to this, zero vectors are added to the beginning of the respective segments such that the left context is guaranteed to be at least 0.5 s long.

6.4.2. Implementation

The implementation follows the implementation for the experiments with the Polish EMA corpus presented in the previous section. The main MATLAB script of this experiment is `runExperimentOnMocha` which is shown in source code listing B.68 on page 411. This initialises the simulation parameters and the input/output settings which are specific to the MOCHA corpus. The simulation proceeds then by calling the main function of the simulation `csmProduction`, shown in listing B.62. The initialization of the memory sequence from the corpus data is implemented in the function `createMemorySequenceFromMocha`, shown in listing B.69 on page 414. This function is similar to the function which creates the memory sequence from the Polish EMA corpus (cf. listing B.64). It produces structurally identical data objects containing the signal data and the segmentation and label information. However, no syllable-related information is extracted since the MOCHA corpus provides only phonetic segment labels and no attempt was made at inferring syllable information from the given unaligned orthographic transcriptions and the phonetic segments.

The remaining functions for this simulation are shared with the previous experiment as they are implemented in a corpus independent way. The details are discussed in section 6.3.2, and the corresponding source code is listed in appendix B.3.1.

6.4.3. Results

The simulation is evaluated essentially in the same way as the first experiment with the Polish data: The model's performance is compared for the three different data types that were used, namely acoustic data vs. articulatory data vs. the combination of acoustic and articulatory data. However, since there is no syllabic information in MOCHA as in the Polish EMA corpus, a syllable-level evaluation is omitted. The *context-accuracy* of the produced segments is measured according to the definitions in equations 6.5, 6.6 and 6.7 on page 229.

Table 6.5 shows the context accuracy results for the data encoded at a sampling rate of 250 Hz. The first observation that can be made is that the absolute accuracy values are much lower in all cases as compared to the results on the Polish data. This is due to the much larger inventory of segment types and the very different structure of the two databases. The Polish EMA corpus was designed for the investigation of a very specific phenomenon. The MOCHA corpus, on the other hand, was designed to provide a broad coverage of the English segment inventory and the coarticulation phenomena

left context accuracy ca^{l}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.082	0.252	0.281	0.224	0.271	0.246
Speaker 2	0.082	0.198	0.272	0.217	0.224	0.234

right context accuracy ca^{r}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.082	0.112	0.111	0.111	0.117	0.111
Speaker 2	0.082	0.104	0.111	0.114	0.116	0.113

left-and-right context accuracy ca^{lr}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.009	0.041	0.045	0.040	0.049	0.044
Speaker 2	0.009	0.032	0.046	0.039	0.045	0.044

Table 6.5.: Context accuracy on English data from the MOCHA corpus (250 Hz) for both audio representations using amplitude envelopes ENV and MFCCs and the respective combined data types with the EMA data.

of continuous speech. Thus, there is a much higher variability in the data set with respect to phone segment combinations. Still, the accuracies are consistently higher than the baseline values for all conditions. One noticeable difference to the results on the Polish corpus is that the articulatory data is not always associated with the highest accuracy values. The accuracy is higher for MFCC data than for the amplitude envelopes. Interestingly, this relation is reversed for the combined data. The combined EMA and envelope data type shows higher accuracy than the combined EMA and MFCC data.

The results for the corpus data encoded at a sampling rate of 500 Hz are shown in table 6.6. These results show the same patterns as the results in table 6.5. When the results in tables 6.5 and 6.6 are compared, it can be seen that the accuracy slightly improves with a higher resolution (i.e. a higher number of frames per second) for the MFCC-encoded acoustic data. This is true for all three contexts: left, right and left-and-right. The results are mixed for the amplitude envelopes and the other data types, however. And

left context accuracy ca^{l}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.082	0.249	0.293	0.223	0.274	0.247
Speaker 2	0.082	0.214	0.280	0.201	0.253	0.221

right context accuracy ca^{r}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.082	0.115	0.116	0.105	0.115	0.106
Speaker 2	0.082	0.109	0.114	0.108	0.119	0.110

left-and-right context accuracy ca^{lr}

	baseline	acoustic		articulatory	combined	
		ENV	MFCC	EMA	ENV+EMA	MFCC+EMA
Speaker 1	0.009	0.044	0.050	0.040	0.048	0.042
Speaker 2	0.009	0.035	0.047	0.035	0.047	0.038

Table 6.6.: Context accuracy on English data from the MOCHA corpus (500 Hz) for both audio representations using amplitude envelopes ENV and MFCCs and the respective combined data types with the EMA data.

the differences are small in all cases which indicates that there is only little effect of sampling rate — at least in this region of 250 Hz and 500 Hz.

6.5. Discussion

I have presented a series of computer simulation experiments in this chapter in which an exemplar-theoretic speech production model, the CSM, is tested on acoustic and articulatory data from a Polish and an English database.

The syllable-level evaluation of the simulations' results shows that they are in line with the findings of Wade et al. (2010, p.229): They show that based on local context information only, syllable-level exemplar selection can emerge. In the experiments presented in this chapter, this syllable-level exemplar selection is indicated by the better-

than-chance context accuracy. Thus, based on local context matching only, segments are selected for a particular syllable environment in the current production stream from a similar syllable environment in memory. The right-context accuracy seems to indicate the presence of some right context information early in the signals in both the acoustic as well as the articulatory domain. This finding supports the conclusions from the experiments discussed in chapter 5 (section 5.4).

6.5.1. Limitations of the current model

There are some problems with the experiments presented in this chapter which I will briefly address here.

The original implementation of the CSM did not explicitly address articulation. A first step towards the integration of articulatory information is presented in this chapter. However, the underlying motor commands and the articulatory gestures are considered rather indirectly through the recorded vocal tract shapes as reflected in the EMA signals. This means that there is no actual motor-planning involved. I do not claim that the dynamically changing vocal tract shapes as measured by EMA represent the articulatory information that is stored in memory. It seems, on the one hand, plausible to assume that articulatory gestures are represented either as intended articulatory movements or as targets of movements — neither of which is directly reflected by EMA recordings. On the other hand, it seems equally plausible to assume (within an exemplar-theoretic framework) that somato-sensory feedback is stored in memory and that it is associated with the corresponding speech items that have been produced. But again, this information is not reflected directly by EMA recordings. Movements of the tongue without any closing contact might be perceived less precisely as the corresponding EMA traces suggest. However, it is important to point out that the stored gestural information which is used in this simulation experiment is real. They represent observed configurations of the vocal tract organs and can be considered "articulatory habits of individual speakers" (Duran et al., 2011b, p.615).

The memory sequence is initialised with all-zeros in the present simulations. The specified left context was sometimes too short for the data from the English database. Those utterances were added to the model's memory sequence such that they are preceded by a stretch of zeros to provide a large-enough left context for the first target segment. These segments might have been simply dropped as targets. Alternatively, the memory sequence could be initialised with a low-amplitude random signal (i.e. noise). This might be a more appropriate representation of silence and it would avoid zero vectors in the context match computations. This was not a problem in the current implementations of the simulation experiments presented in this chapter, but having zero vectors might cause severe problems if another definition of "similarity" is chosen, for example.

The combined data was created by a simple concatenation of the acoustic dimensions and the articulatory dimensions, thus creating vectors with twice the number of dimensions as the uni-modal data types. While it might be plausible to assume that multi-dimensional, multi-modal representations of speech in memory can, in fact, be treated as concatenations of parallel modalities, this poses some methodological problems to the simulation. The three data types are not of the same dimensionality which might affect data processing.

Another inequality between the representations of acoustic and articulatory data is the fact that the acoustic data has been decomposed and the articulatory data has not. The representation using amplitude envelopes in different frequency bands is motivated on auditory grounds. It is not only assumed to contain sufficient information to construct intelligible speech but also to be an adequate representation of (some of) the actual information processed in human speech perception (cf. Wade et al., 2010, p.232). The MFCC-based representation is motivated by the large body of research on speech processing that employs this encoding for speech signals. It is specifically designed for the normalisation of speaker-specific characteristics of the speech signal and faithful representation of linguistic information. How articulatory data as represented by EMA traces might be decomposed in a similar way is not clear — or, if any decomposition is necessary or whether such a decomposition of individual EMA measurements is at all possible in a meaningful way.

The current implementation of the CSM as presented in this chapter does not address some important aspects of the speech *perception–production loop*. The selection process specifies one speech item out of a set of stored candidate exemplars as a production target. While the experiments presented here demonstrate that articulatory information can be readily integrated with the model's sequential memory to be exploited by the selection process, the articulatory processes themselves are still not addressed. Modelling the process which maps a stored item to a motor plan for production is necessary in order to integrate a variety of speech perception/production phenomena with the CSM. The current implementation of the CSM does, for example, not explain the production of *new* speech items. Attempts at producing speech items in a new language for which there is no appropriate ego exemplar must be based on perceived exemplars from other speakers. These exemplars from other speakers are primarily auditory. This new language can be the learner's first language or any other language acquired after that. Even if it is assumed that speech perception is a multimodal (Rosenblum, 2008) or amodal (Fowler, 2004) phenomenon, it can still be assumed that a speaker's own speech productions are perceived, represented and memorised in a different way than speech productions of other speakers. This distinction between "internal" and "external" speech sources is not addressed by the current implementation of the CSM and its interpretation presented in this chapter (cf. the correspondence problem as discussed by Duran et al., 2010b, p.133ff).

6.5.2. Possible future extensions of the Context Sequence model

It is necessary to integrate speech perception and production. Wade and Möbius (2007) modelled speaking rate effects in speech perception, and Wade et al. (2010) modelled (syllable) frequency effects and the importance of surrounding context information in speech production. A complete, exemplar-theoretic account of speech perception and production necessarily needs to implement a closed perception–production loop. Some extensions of the CSM might be outlined as follows.

Category formation and full integration with the perception–production loop

Simulation experiments could test how the CSM can be integrated in a complete perception–production loop. A speech perception/production model like ACT, for example, could provide a suitable framework (cf. section 2.4.4 on page 57). A model like ACT, implemented with an articulatory speech synthesizer, would provide the missing low-level production module. A phonetic map implemented by a SOM does not exclude the existence of a sequential exemplar memory. As Pierrehumbert (2001, p.139) notes, "the labels constitute a level of representation in their own right". Recognition could involve the following steps in such an integrated model: (1) locating the regions of the phonetic map activated by the input; (2) following the links from the phonetic map to the exemplar memory; (3) activation of labels associated with the memory traces; (4) categorisation of the input according to most probable label(s). Production could involve (1) activation of a linguistic category; (2) finding of memory traces associated with the respective label; (3) activation of stored articulatory gestures — i.e. the SOM might not be necessary in production planning but only used to categorise the input and verify the productions. In summary, the interactions between a continuous memory sequence as postulated by the CSM and a dimensionality-reducing cognitive map need to be investigated. Both structures have proven useful in speech perception/production models, as discussed in this thesis.

Higher-level organisation and linguistic abstraction

The CSM does not explicitly address linguistic abstractions, and in fact, the main argument in exemplar theory is that assuming explicit abstractions might not be appropriate in models of cognition. However, linguistics (and maybe language) seems to require abstractions and higher-level organisations. Further computer simulation studies on exemplar-theoretic models can provide insight into the question which abstractions can be described as emergent properties of exemplar representations and which abstractions are explicitly employed in speech and language production and understanding.

The actual implementations of the CSM (those presented here as well as the earlier ones by Wade et al., 2010) only incorporate linguistic abstractions as far as they refer to

the type labels of segments. Lewandowski (2012, p.104f) discusses the relation between episodic and abstract memory representations and the processes involved in exemplar selection. Walsh et al. (2010) propose an exemplar-theoretic model which operates on multiple levels of representation. It does, however, not consider the auditory/phonetic level of linguistic representation which the CSM addresses.

One necessary extension to the CSM would be an account of higher-level linguistic organisation, either through additional layers of increasing linguistic abstraction or through additional mechanisms allowing for various sizes of items that might correspond to phonetic segments, entire syllables, word or phrases.

6.5.3. Conclusion

An extension to the Context Sequence Model (CSM) was presented in this chapter. The simulation experiments investigate the incorporation of articulatory information into the CSM's candidate selection process. As a proof-of-concept, it has been shown that articulatory information can be processed in the same way as acoustic information in a speech production model. The experiments show how both articulatory and acoustic information can be employed to specify a production target exemplar in context.

Some possibilities of future work are outlined above. A self-organising map is a suitable tool for simulation experiments on the emergence of categories. The long, continuous memory sequence of the CSM might provide a suitable tool for simulation experiments on the emergence of segmental information (cf. chapter 4). Combining both in an integrated model of a complete perception–production loop seems like a necessary further development.

7. Discussion and future directions

This final chapter concludes my dissertation. Section 7.1 summarises the addressed research questions and recapitulates the issues addresses in each chapter. Section 7.2 lists the contributions of this thesis. The final section 7.3 gives a global summary and provides a perspective on possible future work.

7.1. Addressed research questions

How can linguistic speech items be recognised and extracted from the continuous and highly variable acoustic input without reference to "innate" linguistic–phonetic constraints, categories or the like?

Do acoustic and articulatory representations facilitate grouping of speech items into similarity-based classes? How do similarity-based groupings of speech items compare to traditional linguistic categories?

How can detailed, continuous acoustic and articulatory representations facilitate speech production in a context-sensitive, exemplar-theoretic model?

What potential applications do computer simulation experiments have in the linguistic study of spoken language? Which research disciplines other than phonetics and phonology offer potential simulation-based approaches to the study of spoken language? What insights from such related fields of research are relevant for computer simulation experiments in phonetics and phonology? And, what is the state of the art in this field and where are its historical beginnings and influences?

Although the experiments presented in chapters 4, 5, and 6 do not directly build on each other, they can be interpreted logically in the order they are presented in this thesis. Speech is a continuous and highly variable medium. In first language acquisition infants face the task of extracting meaningful information from the continuous speech stream. One important part of this learning process is the segmentation of speech into units. The ability to segment speech utterances into smaller chunks might first depend on bottom-up auditory information only. Thus, a first, pre-linguistic segmentation could build the basis for later developments. This issue is addressed in detail in chapter 4. Once speech can be segmented into useful or meaningful units, these units need to

be grouped such that categories can be established. It is assumed that such groupings are performed based on mutual similarities of equally sized speech items. Underlying this assumption are basic ideas of exemplar theory, which are implicitly incorporated in the computer simulations. Resting on the assumption of a perception–production loop in speech perception and production, the similarities of speech items can refer to auditory or articulatory representations or both. The similarity of two given speech items might depend on the modality. Moreover, systematic distinctions between speech items based on their mutual similarities might not necessarily correspond to traditional phonetic or phonological categories. This issue is addressed in chapter 5. Based on observations of systematic variability in fine phonetic, i.e. non-phonemic, details of speech, it can be assumed that speech production employs a rich memory of category representations. Usage-based and exemplar-theoretic approaches postulate that large numbers of experienced speech items are stored in memory. These remembered instances facilitate categorisation of perceived speech by determining the similarity of a stimulus with the exemplars in memory. It also facilitates production of utterances according to the conventions of the speech community. The postulated perception–production loop requires the consideration of both auditory and articulatory representations of remembered speech items (and probably information from other modalities like vision). A model of speech production thus needs to incorporate this multi-modal information. Modelling the sequential production of speech items needs to consider the fact that speech is highly contextual. Context effects might depend on the representation modality of the speech items. This issue is addressed in chapter 6.

Computer simulation experiments allow for a different approach to the study of spoken language. They offer a high degree of control over the data and the experimental environment but also require a higher level of abstraction. Computer simulation experiments can be seen as instances of both theory-driven and data-driven approaches. Using Kohonen's (2001, p.80) terminology, it can be said that computer simulations in phonetics and phonology are concerned with the 'anatomy' of spoken language, while applications in speech technology or NLP are concerned with the 'functional structure' of spoken language. Simulations can complement empirical studies and they can be applied to conduct meta-analyses. The work presented in this thesis — both the discussed literature as well as my own experiments — is representative of a specific approach to the study of spoken language. This thesis addresses the issue of simulation technology in the study of human speech. In particular, chapters 2 and 3 address the questions of what phenomena and aspects of speech can be investigated by means of computer simulation experiments and what specific methods can be used. The historical developments are addressed as well as various examples from phonetics and phonology and related research disciplines. A series of my own computer simulation experiments is presented in chapters 4, 5, and 6, with the corresponding source code of the implementations listed in appendix B.

7.2. Contributions of this thesis

This is probably the first comprehensive review of computer simulation studies in the fields of phonetics and phonology as a whole. Owing to this high level of investigation of the subject and for the sake of brevity, not all of the cited works could be discussed in due detail. However, a comprehensive review of the applications of computer simulation experiments on various aspects of spoken language is given. Chapter 2 provides a historical overview and traces the beginnings of computer simulation experiments in phonetics and phonology back to mechanical and analogue, electronic devices for the production and imitation of speech sounds. Two major influences and sources of related work are identified outside the phonetics and phonology community. First, influences from computer science are identified and reviewed, especially from early considerations on artificial intelligence and work on machine learning. Second computational psychology and cognitive sciences as well as neuro-sciences are identified as providing a relevant body of computer-simulation-based research. Chapter 3 provides a review on relevant methodological issues. Detailed reviews of related work are also provided in chapters 4, 5, and 6 for the different topics addressed by the respective experiments presented in those chapters.

The contributions of chapter 4 can be summarised as follows:

An unambiguous alignment method and definition of true positives is proposed for the classic evaluation measures in the domain of speech segmentation. This addresses frequently found inconsistencies and ambiguities in reports of segmentation evaluations.

A detailed meta-evaluation experiment is presented on the comparison of different external evaluation measures for speech segmentation. This demonstrates a systematic methodology and it provides a detailed investigation of the properties of a number of different evaluation measures.

Two evaluation measures S' and $\widetilde{F}_1$ are proposed which show superior properties over the most commonly used measures like over-segmentation or the classic F_1-measure.

WindowDiff, a measure originally designed for text segmentation, is identified as an appropriate measure for the task of speech segmentation.

A simulation experiment on unsupervised speech segmentation is presented. The results of this experiment refute some expectations about the emergence of segmental structure based on direct comparisons of speech items.

The contributions of chapter 5 can be summarised as:

Two different methods of constructing an approximate confusion matrix for the evaluation of clustering solutions are proposed.

It is shown that continuous EMA as well as acoustic speech data provide useful information about phonetic segment identity in a frame-by-frame clustering scenario. In particular, evidence is found that articulatory information facilitates clustering of consonants more than vowels. Acoustic information, on the other hand, facilitates clustering of vowels more than consonants.

Evidence is found for the presence of right context information in vowel segments spanning over phonological boundaries. Vowels followed by word final obstruent-sonorant clusters can be distinguished in a frame-by-frame clustering from vowels followed by word final obstruents. Despite the phonological analysis of the respective sonorants as extra-syllabic, an influence on the preceding vowel is found.

Finally, the contributions of chapter 6 can be summarised as follows:

An extension of the Context Sequence Model (CSM) is presented which incorporates articulatory information. The performance of the model is shown to be comparable on different representation types.

As a proof of concept, it is shown that continuous EMA recordings can be processed in a speech production model in the same way as continuous acoustic data. On a more abstract level: it is shown that articulatory and acoustic information can be processed in parallel and by the same methods in a context-sensitive speech production model.

Articulatory information represented by EMA recordings is shown to facilitate specification of candidate exemplars in context more than acoustic information does.

Comparable performance (in terms of the applied quality measures) is observed for MFCC representations over amplitude envelopes in the context of the CSM.

An evaluation measure for context accuracy of a sequence of speech segments produced by the CSM is proposed.

7.3. Summary and outlook

I understand this entire thesis as providing a perspective for future work in phonetics and phonology. Computer simulation experiments provide a means of investigating models and testing hypotheses. They can help in discovering unnecessary assumptions and parameters. Simulations can produce unintuitive results which provide further insight into the properties of a model. Computer simulation experiments can address empirically inaccessible phenomena. A researcher devising a computer simulation experiment for the study of speech and language should be familiar with the solutions that more than fifty years of research into artificial intelligence and machine learning have to offer. Thus, such experiments can be performed only by a skilled phonetician with a strong background in computational linguistics and computer science, or in close interdisciplinary collaboration between linguistics and computer scientists.

> “There has been much heated debate over what the subject matter of linguistics *really is*, and what categories of data are permitted to bear on it. A distinction is made between *linguistic evidence* that is appropriate for *linguistics*, versus *psychological* and other evidence that is not. Such discussions, which can be found in all the relevant disciplines, are foreign to naturalistic inquiry. An empirical observation does not come with a notice ‘I am for *X*’, written on its sleeve, where *X* is chemistry, linguistics, or whatever. No one asks whether the study of a complex molecule belongs to chemistry or biology, and no one

> should ask whether the study of linguistic expressions and their properties belongs to linguistics, psychology, or the brain sciences.”
> (Chomsky, 1995, p.33)

The present thesis incorporates a lot of work from disciplines other than phonetics and phonology. Theoretical frameworks and methodologies from fields such as computer science, machine learning and artificial intelligence, or computational psychology and cognitive science have been presented. This thesis might therefore be regarded as interdisciplinary work. Kello and Plaut (2004, p.2356) discuss the DIVA model (cf. section 2.4.2) and one of their earlier simulations of speech acquisition and production and they point out that these “theory-driven models” were based on simplifying assumptions about the acoustic and articulatory speech data representations. They argue that if such simulations fail when applied to more realistic and more complex speech representations, it has to be asked whether the failure is caused by a flaw in the underlying theory or whether the failure is caused by shortcomings of the implementation. I have argued in this thesis that this is one of the major benefits to be gained from computer simulation experiments. Negative results of implemented computational models offer valuable insights for the study of the simulations' underlying theoretical assumptions.

With respect to the experiments presented in the second part of this thesis, some specific possibilities for future work are as follows.

The whole-sequence segmentation approach to speech segmentation presented in chapter 4 can be tested on artificial data with more speech-like properties, like random noise representing silence or construction of the corpus from repeated pseudo-segments according to different distributions.

The results of the vowel clustering experiment presented in section 5.4 show that long-distance coarticulatory information is present in phonetic segments. This finding is important for the whole-sequence segmentation approach presented in section 4.4. It cannot be assumed that a similarity function will show considerable discontinuities corresponding to linguistic boundaries. One reason is the presence of segmental context information across segment boundaries. The study by Johnson (1997b, p.107) showed that right context information exist in onset syllables of poly-syllabic words. The vowel clustering experiments in section 5.4 confirm this general observation of long-distance influences. The results show that an equivalent effect can be observed on individual vowel segments which indicate the presence or absence of an extrametrical coda consonant.

The whole-sequence segmentation experiments section 4.4 are based on exemplar-theoretic considerations. In particular, the memory representation and the speech processing process as outlined in the CSM is one of the basic assumptions on which this experiment is based.

The application of fuzzy clustering methods needs to be investigated to account for the high variability and the apparent high degree of ambiguity in speech data and

the continuous nature of boundariness. This would also require a different approach. Phonetics and phonology need to consider the ubiquitous introduction of variation in perception and production due to varying individual properties of the language users and varying environmental or contextual conditions. Language is a self-organising phenomenon, embodied in the analogue human brain which operates highly parallel, asynchronous and associatively. Although language knowledge often appears to be categorical, it is at the same time also gradient and not binary at all. It seems more appropriate to use real numbers instead of integers in the description of language and probabilities instead of rules.

In addition to the modified F-measure, the proposed relative TP can be applied in other evaluation measures which are based on the total number of true positives. The properties and the utility of such measures remains to be investigated as well as the performance of the discussed evaluation measures on higher segmentation levels, for example in syllable, word or utterance segmentation tasks.

The computer simulation experiments presented in chapter 5 require further attention. The observed results of the first experiment on clustering phonetic segments need to be verified by additional experiments on more data. It would be interesting to apply the clustering method to a data set covering the full phonetic inventory of a language with articulatory and acoustic recordings. Different languages need to be examined. The observed results of the vowel clustering could be further investigated by empirical experiments, testing whether the found discriminability of the vowels can be perceived.

The problem of speech segmentation is addressed in chapter 4 and chapter 5 addresses the problem of speech item classification. In machine learning, a classification task can be defined as finding a mapping from a set of input items X to a set of labels Y. Applied to the speech domain in language acquisition, the problem is not only that we don't know the optimal set of labels Y for speech data X, we also do not know the optimal representation of X itself.

The computer simulation experiments presented in chapter 6 operate on a data representation combining acoustic and articulatory information. The applied scheme puts equal weights on both data types. Different weights associated with different modalities should be investigated as well as different time scales. The slowly varying articulatory information could be processed with a different time resolution, resp. on a different scale than the acoustic information.

A detailed review of possible extension to the CSM is given in section 6.5.2. It needs to incorporate the full perception–production loop.

In summary, computer simulations are great! They offer vast possibilities for the study of speech and our investigation of all of its aspects. I borrow my conclusion from one of the greatest scientists:

> "We can only see a short distance ahead, but we can see plenty there that needs to be done."
> (Turing, 1950, p.460)

Appendix

Appendix A. Databases

A.1. Polish EMA corpus

The Polish EMA corpus is a database of controlled utterances which was originally designed to investigate voicing phenomena and the *C-centre effect* in Polish (Mücke et al., 2010; Bruni, 2011). The C-centre effect is a phenomenon related to segmental syllable structure. More precisely, it is an effect of consonant coordination in consonant clusters.

The corpus contains audio recordings and time-aligned articulatory measurements obtained through electromagnetic midsagittal articulography (EMA) using a CARSTENS AG100, Electromagnetic Articulograph with 10 channels. Signals from four sensors were used for the simulation experiments: two for the tongue body movements (recorded traces of sensors placed 3 cm and 4 cm behind the tongue tip), one for the tongue tip and one for the lip distance (from two sensors placed on the vermillion border of the upper and lower lip).

Three adult, non-professional, native speakers of Polish were recorded (two female and one male) producing a set of target words embedded in carrier phrases. The target words were selected such that they represent real Polish words. They contain the target syllables of the original investigation of the C-centre effect. The details of this original motivation for the construction of the corpus are actually not relevant to the computer simulation experiments discussed in this thesis as it was a laboratory study with human subjects, and the questions addressed with the simulation experiments were different. However, in order to understand the particular structure and the contents of a given corpus, it is necessary to know its original purpose.

The target words are listed in table A.1. Each of the words was recorded in a carrier phrase that provides identical articulatory contexts for all target consonants and consonant clusters. The target consonant/cluster is always preceded by a high vowel (/i/ or /ɨ/) and followed by a low vowel (/a/), either within or across word boundaries. These are the two carrier phrases (the target word example is printed in bold face):

onset: *"Ona mówi **pranie** aktualnie"*
(*She is saying **laundry** currently*)

coda: *"Ona powiedziała **Cypr** aktualnie"*
(*She said **Cyprus** currently*)

The target words were produced in two different conditions: with and without emphasis. Utterances that could not be processed automatically (due to inconsistent labelling or some missing or incomplete signal files etc.) have been excluded. The resulting two splits of the corpus comprise 336 (3×112) utterances in the emphasis part, labelled "emph" in the text, and 337 ($113 + 2 \times 112$) utterances in the non-emphasis part, labelled "noemph". Only utterances from the emph part were used for the experiments presented prior to this thesis (Duran et al., 2011b,a,c, 2012b,a).

C target	onset	coda
p	padnij	typ
k	kadisz	tik
l	labrys	gil
r	rabin	tir
p + l	plamić	ZUPL
p + r	pranie	Cypr
k + l	klawisz	cykl
k + r	krasić	WIKR

Table A.1.: Target words containing the target consonants from the Polish EMA corpus.

All the target words were manually segmented and labelled at the word level and at the phonetic level. Only the segments of the target syllables are annotated, i.e. the consonants shown in table A.1 and the corresponding vowels /i, ɨ, u, a/. Additionally, gestural events ("onset", "target" and "offset" of articulatory gestures) were labelled for the tongue body, the tongue tip and the lip distance at three distinct layers to allow the annotation of overlapping gestures of the various articulators.

The original corpus data was annotated in the EMU Speech Database System[1] and stored in Wave and Simple Signal File Format (SSFF). Prior to any of the experiments

[1] Available online at: `http://emu.sourceforge.net`. See also the corresponding manual by Cassidy (2004).

described in this thesis, the data has been extracted and stored in binary `.mat`-files to allow fast access and fast automatic processing of the entire corpus in MATLAB. File formats which are designed to be opened and manipulated via graphical user interfaces are not necessarily optimised for fast access and processing by computer programs. Since not all of the data from the corpus was used for all experiments, the amount of data to be loaded for each run of an experiment could be significantly reduced by pre-processing it.

A.2. IMS Unit Selection Corpus

The database referred to as the IMS unit selection corpus in this thesis is a speech database which was originally developed for unit selection speech synthesis for a dialogue system (Barbisch et al., 2007, p.304; extending earlier work by Schweitzer et al., 2006).

The corpus contains two hours of speech recordings from a male speaker and three hours of speech recordings from a female speaker. The sub-corpora are labelled as "IMS:ms" for the male speaker and "IMS:rk" for the female speaker. Both speakers are professional, adult native speakers of standard German. The corpus is labelled at the segment, syllable and word level and it contains prosodic annotations. The segment types available in the corpus and the used labels are shown in table A.2.

The content of the recorded utterances covers the following domains: soccer reports, tourist information, newspaper articles, dialogue system prompts and numbers (Schweitzer, 2010b, p.115f). The corpus contains a considerable amount of proper names. One target application was an information dialogue system for the 2006 FIFA World Cup. In order to produce a correct pronunciation for foreign names, some material with non-German segments is contained in the corpus (Barbisch et al., 2007, p.307). This can be seen in the table which contains symbols that usually cannot be found in a standard German phone inventory.

label	IPA	example
2	ø	zusammeng**e**rührt
2:	øː	erh**ö**hen
3:	ɜː	F**er**dinand
6	ɐ	To**r**
9	œ	köpft
<P>	—	*(silence)*
@	ə	vorb**e**halt**e**n
C	ç	s**ich**
D	ð	the
E	ɛ	Z**e**ntrum
E:	ɛː	w**ä**re
I	ɪ	sich
L	lˠ	E**ll**iott
N	ŋ	li**n**ks
O	ɔ	Rostock
O:	ɔː	Shaun
OY	ɔ͡ɪ	d**eu**tlich
Q2:	ʔøː	**ö**kologisch
Q9	ʔœ	**ö**ffentlich
Q@	ʔə	**A**merican
QE	ʔɛ	**e**rwischt
QE:	ʔɛː	**ä**hnlich
QI	ʔɪ	in
QO	ʔɔ	**O**tto
QO:	ʔɔː	**Au**rtis
QOY	ʔɔ͡ɪ	**Eu**ropa
QU	ʔʊ	um
QY	ʔʏ	**ü**ppiger

label	IPA	example
Qa	ʔa	als
Qa:	ʔaː	aber
QaI	ʔa͡ɪ	**ei**n
QaU	ʔa͡ʊ	**au**s
Qa~	ʔã	**En**gagement
Qe:	ʔeː	erst
QeI	ʔe͡ɪ	**A**drian
Qe~	ʔẽ	**In**terieur
Qi	ʔi	**i**soliert
Qi:	ʔiː	ihn
Qo:	ʔoː	oder
QoU	ʔo͡ʊ	open
Qu	ʔu	unbefristet
Qu:	ʔuː	Uwe
Qy:	ʔyː	**ü**ber
R	ʁ	rechts
S	ʃ	**Sch**weiz
T	θ	Smith
U	ʊ	durch
Y	ʏ	fünf
Z	ʒ	**Ž**ivković
a	a	das
a:	aː	nach
aI	a͡ɪ	w**ei**l
aU	a͡ʊ	k**au**m
a~	ã	Ch**an**ce
b	b	bald
d	d	die

label	IPA	example
e:	eː	der
eI	e͡ɪ	volley
e~	ẽ	Souterr**ain**
f	f	für
g	g	gegen
h	h	**h**och
i	i	Afrika
i:	iː	d**ie**
j	j	jeder
k	k	Konzert
l	l	mal
m	m	dem
n	n	nach
o	o	HipHop
o:	oː	Tor
oU	o͡ʊ	H**o**mepage
o~	õ	Sais**on**
p	p	Pause
r	r	Broadway
s	s	es
t	t	Tal
u	u	M**ou**stapha
u:	uː	zu
v	v	**w**ar
w	w	Air**w**ays
x	x	na**ch**
y:	yː	Torh**ü**ter
z	z	sich

Table A.2.: The phonetic segment labels used in the IMS unit selection corpus with their corresponding IPA symbols and example words containing the respective segment taken from the corpus (segments printed in bold face if necessary).

A.3. Polish BOSS Corpus

The Polish BOSS corpus is a unit selection speech database developed for the Bonn Open Synthesis System[2] (Demenko et al., 2008a,b). The corpus contains recordings of a professional male speaker and covers a wide range of consonant clusters, di- and triphones and the most frequent lexical items (Demenko et al., 2008b, p.86f). Based on the phonetic transcription, the speech data was segmented automatically and manually corrected (Demenko et al., 2008a, p.1650). Table A.3 shows the segment labels used in the corpus (excluding prosodic and boundary annotations; cf. Demenko et al., 2008b, p.87f, or Bachan, 2007, p.25ff).

label	IPA	example
J	ɟ	**g**itarze
N	ŋ	ba**n**k
S	ʃ	**sz**częśliwie
Z	ʒ	**ż**e
a	a	auta
b	b	bokser
c	c	**k**iedy
d	d	Daniel
d^Z	d͡ʒ	**drz**azgi
d^z	d͡z	**dz**ban
d^z'	d͡ʑ	**dzi**ękuję
e	ɛ	ekskluzywnie
f	f	film
g	g	gra
i	i	i
j	j	jednego
j~	j̃	część
k	k	kupi
l	l	lubie

label	IPA	example
m	m	moja
n	n	na
n'	ɲ	**ni**e
o	ɔ	ofiarę
p	p	ponieważ
r	r	rozbrzmiewa
s	s	sobie
s'	ɕ	**ś**cian
t	t	to
t^S	t͡ʃ	**cz**ubek
t^s	t͡s	**c**oś
t^s'	t͡ɕ	**ci**e
u	u	urzędnik
v	v	niedź**w**iedź
w	w	**ł**uk
w~	w̃	dużą
x	x	**ch**leba
y	ɨ	tylko
z	z	znam
z'	ʑ	jeździ

Table A.3.: The phonetic segment labels used in the Polish BOSS corpus with their corresponding IPA symbols and example words containing the respective segment taken from the corpus (segments printed in bold face if necessary).

[2] http://www.sk.uni-bonn.de/forschung/phonetik/sprachsynthese/boss (accessed 2012-12-22).

A.4. MOCHA corpus

The MOCHA (Multi-CHannel Articulatory) database, also called MOCHA-TIMIT, contains phonetically annotated recordings of Southern English speech of two native speakers: one male, labelled with the ID "msak", and one female, labelled with ID "fsew" (Wrench, 1999). The corpus offers acoustic and articulographic speech recordings (esp. EMA measurementes which were of particular interest in the experiments presented in this thesis). It was designed to serve as training data for speaker-independent automatic speech recognition systems. For this purpose, it was originally planned to record a larger number of speakers (cf. Wrench, 2000, p.7). A total of 460 short sentences (some of which are taken from the TIMIT database) was recorded to provide a phonetically balanced set with broad coverage of coarticulation phenomena in English.

Articulographic recordings were made using a Carstens AG100 Electomagnetic Articulograph. The sensors were placed at the vermillion border of the upper and lower lips, the lower incisor (for jaw movement), the tongue tip, the tongue blade and dorsum and the soft palate (Wrench, 2000, p.7). The recorded EMA data is stored in binary "track files" as provided by the Edinburgh Speech Tools Library (King et al., 2003)[3] at a resolution of 16 bit, with a sampling rate of 500 Hz (i.e. 500 frames per second, or a frame shift of 2 ms). The audio data is stored in uncompressed binary files at 16 bit resolution, with a sampling rate of 16 kHz.

The original corpus files have been pre-processed for the experiments presented in chapter 6 to allow faster access for the various Matlab and Java scripts. The articulatory data was extracted from the EMA files and converted to plain ASCII with the following Linux shell commands:[4]

```
foreach f (./*.ema)
ch_track $f -info > $f:t:r.ema.info
ch_track $f -o $f:t:r.ema.ascii
end
```

The original audio files could not be opened in Matlab. Therefore, the audio files were converted to `snd` format, which is readable with Matlab's `auread` function. The shell command used to process the audio files is:

```
foreach f (./*.wav)
ch_wave $f -info > $f:t:r.wav.info
ch_wave $f -otype snd -o $f:t:r.wav.snd
end
```

[3] The library is available online at http://www.cstr.ed.ac.uk/projects/speech_tools/ (accessed: 2011-09-13)

[4] The tools `ch_track` "Track file manipulation" and `ch_wave` "Audio file manipulation" are part of the Edinburgh Speech Tools Library, documented online in the library's manual (King et al., 2003, chapter 3).

From these converted files, another set of pre-processed binary MATLAB files was created for the simulation experiments such that the data can be loaded quickly and efficiently.

The phonetic segment labels used in the MOCHA corpus are listed in table A.4. The MOCHA database is annotated only at the phonetic segment level. For each utterance, an orthographic transcription is also provided, but this is not aligned with the respective signals of the recordings, and it is not used in the experiments presented in this thesis.

label	IPA	example
a	æ	can
e	ɛ	yell
i	ɪ	this
ii	iː	thieves
iy	i	only
@	ə	over
@@	ɚ	thirty
uh	ʌ	us
aa	ɑ	hard
o	ɒ	on
oo	ɔ	more
u	ʊ	good
uu	u	you
ai	aɪ	by
ei	eɪ	may
eir	ɛə	their
i@	ɪə	near
oi	ɔɪ	oil
ou	oʊ	roll
ow	aʊ	found

label	IPA	example
b	b	beg
d	d	do
g	g	grades
p	p	pick
t	t	to
k	k	cat
ch	ʧ	chocolate
jh	ʤ	jewels
f	f	for
h	h	hard
s	s	safe
sh	ʃ	she
th	θ	wealth
v	v	over
z	z	easy
zh	ʒ	prestige
dh	ð	this
l	l	silly
r	ɹ	roll
w	w	worry
y	j	yell
m	m	money
n	n	can
ng	ŋ	swing
breath		*(breath)*
sil		*(silence)*

Table A.4.: The phonetic segment labels used in the MOCHA corpus with their corresponding IPA symbols and example words containing the respective segment taken from the corpus (segments printed in bold face if necessary).

Appendix B. Source code

The file name suffixes of the source code listings indicate the programming language as follows:

.java	Java code
.m	Matlab code
.c	C code
.R	R code

B.1. Segmentation

B.1.1. Evaluation diagnostics

This sections lists the Java source code for the experiment on the comparison of evaluation measures for speech segmentation (which is presented in section 4.3 on page 110).

Listing B.1: RunSegmentationDiagnostic.java

```
package de.uniStuttgart.danielDuran;

import java.io.File;
import java.nio.charset.Charset;
import java.util.Locale;
import de.uniStuttgart.danielDuran.evaluation.Comparison;

/**
 * @author Daniel Duran
 */
public class RunSegmentationDiagnostic {

   public static void main(String[] args)
   {
      // Check and read program arguments
      if(args.length != 7){
         printHelp(); return;
      }

      // argument args[0]: input file
      File in = new File(args[0]);
```

```
22      if( ! in.exists()){
23         printHelp(); return;
24      }
25
26      // argument args[1]: input frame rate
27      int frameRate = Integer.parseInt(args[1]);
28      if(0 > frameRate){
29         printHelp(); return;
30      }
31
32      // argument args[2]: input file encoding
33      String inputEnc = args[2];
34      if( ! Charset.availableCharsets().containsKey(inputEnc) ){
35         System.err.println("Specified␣invalid␣character␣encoding!");
36         printHelp(); return;
37      }
38
39      // argument args[3]: output directory
40      File outBaseDir = new File(args[3]);
41      if( ! outBaseDir.exists() ){
42         System.out.println("Creating␣output␣directory:␣" + outBaseDir.↩
            getAbsolutePath() );
43         outBaseDir.mkdirs();
44      }
45
46      // argument args[4]: tolerance window sizes in ms
47      String[] tolStr = args[4].split(",");
48      if(tolStr.length < 1){
49         printHelp(); return;
50      }
51      double[] tolerances = new double[tolStr.length];
52      for(int i=0; i<tolStr.length; i++){
53         tolerances[i] = Double.parseDouble(tolStr[i]);
54         if(0 > tolerances[i]){
55            System.err.println("Tolerance␣window␣size␣must␣not␣be␣negative!");
56            printHelp(); return;
57         }
58      }
59
60      // argument args[5]: alpha values for baseline segmenters
61      String[] alStr = args[5].split(",");
62      if(alStr.length < 1){
63         printHelp(); return;
64      }
65      double[] alphas = new double[alStr.length];
66      for(int i=0; i < alStr.length; i++){
67         alphas[i] = Double.parseDouble(alStr[i]);
68         if(0 > alphas[i]){
69            System.err.println("Alpha␣factor␣must␣not␣be␣negative!");
```

```
70            printHelp(); return;
71         }
72      }
73
74      // argument args[6]: number of repeated runs for the random baseline
75      int runs = Integer.parseInt(args[6]);
76      if(0 > runs){
77         System.err.println("Number␣of␣runs␣must␣not␣be␣negative!");
78         printHelp(); return;
79      }
80
81      System.out.println("Start:␣␣␣Data␣file="+in.getAbsolutePath());
82      Locale l = Locale.UK;
83      CorpusReader c = new CorpusReader(in, inputEnc, "", frameRate);
84
85      // now do segmentations for all tolerance window sizes:
86      for(int tx=0; tx<tolerances.length; tx++){
87         System.out.println("Start:␣Tolerance="+tolerances[tx]);
88
89         File outDir = new File(outBaseDir,
90               in.getName() + "/delta"+String.format("%.0f",tolerances[tx]) );
91         System.out.println("Start:␣Output␣dir="+outDir.getAbsolutePath());
92
93         Comparison comp = new Comparison(outDir,l);
94         comp.printToleranceWindowStatistics(c, tolerances[tx]);
95         comp.runDiagnosticSegmenters(c, tolerances[tx]);
96
97         for(int ax=0; ax<alphas.length; ax++){
98            System.out.println("Start:␣Alpha="+alphas[ax]);
99            File outDirAlpha = new File(outDir, "alpha"+String.format(l,"%.2f",↩
                 alphas[ax]));
100           comp = new Comparison(outDirAlpha,l);
101           comp.runBaselineConstSegmenter(c, tolerances[tx], alphas[ax]);
102           comp.runBaselineRandSegmenter(c, tolerances[tx], alphas[ax], runs);
103        }
104     }
105     System.out.println("done.");
106  }
107
108  private static void printHelp()
109  {
110     System.out.println("Usage:␣");
111     System.out.println("␣required␣arguments:");
112     System.out.println("␣(1)␣Input␣file.");
113     System.out.println("␣(2)␣Input␣frame␣rate.");
114     System.out.println("␣(3)␣Input␣file␣encoding.");
115     System.out.println("␣(4)␣Output␣base␣directory.");
116     System.out.println("␣(5)␣Tolerance␣window␣sizes␣in␣ms:");
117     System.out.println("␣␣␣␣A␣comma␣separated␣list␣of␣float␣values.");
```

```
118         System.out.println("␣␣␣␣␣Example:␣10.0,20.0␣");
119         System.out.println("␣(6)␣Scaling␣factor␣alpha␣for␣the␣baseline");
120         System.out.println("␣␣␣␣␣segmenters:␣A␣comma␣separated␣list␣of␣");
121         System.out.println("␣␣␣␣␣float␣values:␣Example:␣0.5,1.0,2.0");
122         System.out.println("␣(7)␣Number␣of␣repeated␣runs␣for␣the␣random");
123         System.out.println("␣␣␣␣␣baseline␣segmenter.");
124     }
125 }
```

Listing B.2: CorpusReader.java

```
1  package de.uniStuttgart.danielDuran;
2
3  import java.io.File;
4  import java.util.Arrays;
5  import de.uniStuttgart.danielDuran.tools.SeqStat;
6  import de.uniStuttgart.danielDuran.tools.TabFileReader;
7
8  /**
9   * @author Daniel Duran
10  */
11 public class CorpusReader implements ICorpusReader {
12
13     private final int[] boundaries;
14     private final int totalFrames;
15     private final int frameRate;
16     private final SeqStat segLengths;
17
18     public CorpusReader(File inFile, String encoding, String separator, int ↩
          frameRate)
19     {
20         this(inFile, encoding, separator, frameRate, 0, Integer.MAX_VALUE);
21     }
22
23     public CorpusReader(File inFile, String encoding, String separator, int ↩
          frameRate, int from, int to )
24     {
25         System.out.println("Reading␣data...");
26         this.frameRate = frameRate;
27         String[][] rc = TabFileReader.getContents(encoding, separator, true, false, ↩
             inFile, from, to);
28
29         if(null!=rc){
30             System.out.println("Found␣"+rc.length+"␣lines␣in␣file␣" + inFile.↩
                getAbsolutePath());
31             double[] ref = new double[rc.length];
32             Arrays.fill(ref, 0.0);
33             String currentLabel = rc[0][0];
34             int bcount = 0;
```

```
35          for(int i = 1; i < rc.length; i++){
36             if(!rc[i][0].equals(currentLabel)){
37                ref[i] = 1.0;
38                currentLabel = rc[i][0];
39                bcount++;
40             }
41          }
42          int[] refIndex = new int[bcount];
43          int ptr=0;
44          for(int i = 0; i < rc.length; i++){
45             if(ref[i]==1.0){
46                refIndex[ptr] = i;
47                ptr++;
48             }
49          }
50          this.boundaries  = refIndex;
51          this.totalFrames = ref.length;
52          // ----------------------------------
53          // compute segment length statistics
54
55          // get segment lengths
56          double[] lengths = new double[refIndex.length+1];
57
58          int lastIndex = 0;
59          for(int i=0; i< refIndex.length; i++){
60             double len = refIndex[i] - lastIndex;
61             lengths[i] = len;
62             lastIndex  = refIndex[i];
63          }
64          // add last segment length (between the last boundary and the end of the ↩
               sequence)
65          lengths[refIndex.length] = refIndex[refIndex.length-1] - lastIndex;
66
67          segLengths = new SeqStat();
68          segLengths.computeStatistics(lengths);
69
70          double lengthMean = segLengths.getMean() / (double)frameRate;
71          double lengthSd   = segLengths.getSD()   / (double)frameRate;
72          double lengthMin  = segLengths.getMin()  / (double)frameRate;
73          double lengthMax  = segLengths.getMax()  / (double)frameRate;
74
75          // print out length statistics:
76          System.out.println("Opened␣data␣from:␣");
77          System.out.println(inFile.getAbsolutePath());
78          System.out.println("Frames:␣␣␣" + this.totalFrames);
79          System.out.println("Segments:␣" + this.boundaries.length );
80          System.out.println("Segment␣length␣statistics:");
81          System.out.println("␣-␣mean␣length:␣"
82                + String.format("%.4f␣s;␣", lengthMean)
```

```
83                + String.format("%,.2f␣frames",segLengths.getMean()));
84          System.out.println("␣-␣sd:␣␣␣␣␣␣␣␣␣"
85                + String.format("%.4f␣s;␣", lengthSd)
86                + String.format("%,.2f␣frames",segLengths.getSD()));
87          System.out.println("␣-␣min.␣length:␣"
88                + String.format("%.4f␣s;␣", lengthMin)
89                + String.format("%,.2f␣frames",segLengths.getMin()));
90          System.out.println("␣-␣max.␣length:␣"
91                + String.format("%.4f␣s;␣", lengthMax)
92                + String.format("%,.2f␣frames",segLengths.getMax()));
93
94       } else {
95          this.boundaries  = new int[0];
96          this.totalFrames = 0;
97          this.segLengths  = null;
98       }
99    }
100
101   public int[] getBoundaryIndices() {
102      return this.boundaries;
103   }
104
105   public int getNumberOfFrames(){
106      return this.totalFrames;
107   }
108
109   public double[][] getData() { return null; }// Not used
110
111   public String[] getLabels() { return null; }// Not used
112
113   public int getFrameRate() {
114      return this.frameRate;
115   }
116
117   public int getNumberOfSegments() {
118      return this.boundaries.length ;
119   }
120
121   public SeqStat getSegmentLengthStats() {
122      return segLengths;
123   }
124 }
```

Listing B.3: ICorpusReader.java

```
1 package de.uniStuttgart.danielDuran;
2
3 import de.uniStuttgart.danielDuran.tools.SeqStat;
4
```

```
5 public interface ICorpusReader {
6
7    /**
8     * @return the internal start indices of the segments, i.e. the first
9     * segment of the corpus starts at 0, and the last segment starts at
10    * the last entry of the returned array.
11    * <p>Index array format:
12    * <pre>
13    * [  segment 1  ][ segment 2][ segment 3]
14    * [-][-][-][-][-][+][-][-][-][+][-][-][-] .length=13
15    * 0  1  2  3  4  5  6  7  8  9  10 11 12
16    * ==
17    * [5][9]
18    * </pre>
19    * Notes:
20    * The start index i is equal to the length of the preceding segment
21    * i-1 minus the start index of segment i-1. The length of the first
22    * segment is equal to the value at i=0; The last segment has length
23    * getNumberOfFrames() minus the last value of this array.
24    */
25   public int[] getBoundaryIndices();
26
27   public int getNumberOfFrames();
28
29   public double[][] getData();
30
31   public String[] getLabels();
32
33   public int getFrameRate();
34
35   public int getNumberOfSegments();
36
37   public SeqStat getSegmentLengthStats();
38 }
```

Listing B.4: TabFileReader.java

```
1 package de.uniStuttgart.danielDuran.tools;
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileInputStream;
6 import java.io.IOException;
7 import java.io.InputStreamReader;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 public class TabFileReader {
12
```

```
13    /**
14     * Read contents from a file into a String array
15     * @param encoding
16     * @param separator
17     * @param ignoreEmptyLines
18     * @param trim
19     * @param in
20     * @return
21     */
22    public static String[][] getContents(
23          String encoding,
24          String separator,
25          boolean ignoreEmptyLines,
26          boolean trim,
27          File in,
28          int from, int to)
29    {
30       List<String[]> lines = new ArrayList<String[]>();
31       int maxColumns=0;
32       BufferedReader br = null;
33       try {
34          int c =0;
35          br = new BufferedReader(new InputStreamReader(new FileInputStream(in), ↩
             encoding));
36          String line = null;
37          while( (line = br.readLine()) != null) {
38             if(c >= from && c <= to) {
39                if(ignoreEmptyLines && line.trim().isEmpty()){
40                   continue;
41                }
42                String[] fields;
43                if(trim){
44                   fields = line.trim().split(separator);
45                } else {
46                   fields = line.split(separator);
47                }
48                if(fields.length>maxColumns)
49                   maxColumns = fields.length;
50                lines.add(fields);
51             }
52             else if(c > to){
53                break;
54             }
55             c++;
56          }
57       } catch (Exception e) {
58          e.printStackTrace();
59       } finally {
60          // make sure the stream is closed:
```

```
61          if(null!=br){
62             try {
63                br.close();
64             } catch (IOException e) {
65                e.printStackTrace();
66             }
67          }
68       }
69       return lines.toArray(new String[lines.size()][maxColumns]);
70    }
71 }
```

Listing B.5: SeqStat.java

```
1 package de.uniStuttgart.danielDuran.tools;
2
3 /**
4  * Simple helper class to compute min, max, mean etc. of a given array
5  * @author Daniel Duran
6  */
7 public class SeqStat {
8
9    private double sum;
10   private double count;
11   private Double sd;
12   private double[] data;
13   private Double min;
14   private Double max;
15
16   public SeqStat(){
17      this.reset(null);
18   }
19
20   private void reset(double[] data){
21      this.data = data;
22      if(null!=data){
23         this.count = (double)data.length;
24      } else {
25         this.count = 0.0;
26      }
27      this.sum   = 0.0;
28      this.sd    = null;
29      this.min   = Double.MAX_VALUE;
30      this.max   = -Double.MAX_VALUE;
31   }
32
33   public void computeStatistics(double[] data)
34   {
35      this.reset(data);
```

```
36      for(int i = 0; i<data.length; i++) {
37         this.sum += data[i];
38         if(data[i] < this.min){
39            this.min = data[i];
40         }
41         if(data[i] > this.max){
42            this.max = data[i];
43         }
44      }
45   }
46
47   public double getSD()
48   {
49      if(null==this.sd && null!=this.data) {
50         double mean = getMean();
51         double sum  = 0.0;
52         for(int i = 0; i < this.data.length; i++) {
53            sum = sum + ( (data[i] - mean) * (data[i] - mean) );
54         }
55         this.sd = Math.sqrt( sum / (count - 1.0) );
56      }
57      return this.sd;
58   }
59
60   public Double getMean(){
61      return this.sum / this.count;
62   }
63
64   public int getElementCount(){
65      return (int)this.count;
66   }
67
68   public Double getMin(){
69      if(this.min < Double.MAX_VALUE)
70         return this.min;
71      else
72         return null;
73   }
74
75   public Double getMax(){
76      if(this.max > -Double.MAX_VALUE)
77         return this.max;
78      else
79         return null;
80   }
81 }
```

Listing B.6: Comparison.java

```
1 package de.uniStuttgart.danielDuran.evaluation;
2
3 import java.io.File;
4 import java.util.ArrayList;
5 import java.util.Locale;
6 import de.uniStuttgart.danielDuran.CorpusReader;
7 import de.uniStuttgart.danielDuran.segmenter.BaselineConst;
8 import de.uniStuttgart.danielDuran.segmenter.BaselineRandNorm;
9 import de.uniStuttgart.danielDuran.segmenter.ISegmenter;
10 import de.uniStuttgart.danielDuran.segmenter.SegmenterCenter;
11 import de.uniStuttgart.danielDuran.segmenter.SegmenterDelta;
12 import de.uniStuttgart.danielDuran.segmenter.SegmenterDeltaMinusOne;
13 import de.uniStuttgart.danielDuran.segmenter.SegmenterDeltaPlusOne;
14 import de.uniStuttgart.danielDuran.segmenter.SegmenterDouble;
15 import de.uniStuttgart.danielDuran.segmenter.SegmenterHalfRef;
16 import de.uniStuttgart.danielDuran.segmenter.SegmenterHalfSegments;
17 import de.uniStuttgart.danielDuran.segmenter.SegmenterPerfect;
18 import de.uniStuttgart.danielDuran.segmenter.SegmenterRelativeBias;
19 import de.uniStuttgart.danielDuran.segmenter.SegmenterShift;
20 import de.uniStuttgart.danielDuran.segmenter.SegmenterTotal;
21 import de.uniStuttgart.danielDuran.tools.TabFileWriter;
22
23 /**
24  * @author Daniel Duran
25  */
26 public class Comparison {
27
28    private final File outDir;
29    private final Locale loc;
30
31    public Comparison(File outputDirectory, Locale l) {
32       this.outDir = outputDirectory;
33       this.outDir.mkdirs();
34       this.loc = l;
35    }
36
37    /**
38     * Wrapper to start the comparison with the diagnostic
39     * segmentation methods.
40     * @param c
41     * @param toleranceInMs
42     */
43    public void runDiagnosticSegmenters(
44          CorpusReader c,
45          double toleranceInMs )
46    {
47       // define diagnostic Segmenters
48       ArrayList<ISegmenter> segmenters = new ArrayList<ISegmenter>();
49       // ------------------------------------------------------------------
```

```
50      segmenters.add(new SegmenterPerfect("\\segRef", c));
51      // ------------------------------------------------------------------
52      // segment at every possible position:
53      segmenters.add(new SegmenterTotal("\\segAll", c));
54      // ------------------------------------------------------------------
55      // use every second reference boundary:
56      segmenters.add(new SegmenterHalfRef("\\segHalf", c));
57      // ------------------------------------------------------------------
58      // use every reference boundary and place an additional boundary
59      // in the middle of each segment:
60      segmenters.add(new SegmenterHalfSegments("\\segAllZ",c));
61      // ------------------------------------------------------------------
62      // for every reference boundary i, set a hypothesized boundary at
63      // j=(i + tolerance + 1 frame):  hyp = ref+delta+1
64      segmenters.add(new SegmenterDeltaPlusOne("\\segDplus", c, toleranceInMs));
65      // ------------------------------------------------------------------
66      // for every reference boundary i, set a hypothesized boundary at
67      // j=(i + tolerance ):  hyp = ref+delta
68      segmenters.add(new SegmenterDelta("\\segD", c, toleranceInMs));
69      // ------------------------------------------------------------------
70      // for every reference boundary i, set a hypothesized boundary at
71      // j=(i + tolerance - 1 frame):  hyp = ref+delta
72      segmenters.add(new SegmenterDeltaMinusOne("\\segDminus",c,toleranceInMs));
73      // ------------------------------------------------------------------
74      // set a boundary at the midpoint of every reference segment:
75      // Hyp_i = (Ref_i-1 - Ref_i) / 2
76      segmenters.add(new SegmenterCenter("\\segCent", c));
77      // ------------------------------------------------------------------
78      // set of every reference boundary at index i a hyp boundary at
79      //   i-1 and i+1: hearst "Problem 2"
80      segmenters.add(new SegmenterDouble("\\segDouble", c));
81      // ------------------------------------------------------------------
82      // hearst "Problem 4"
83      // (9) should be penalized more than (10)
84      // shift every reference boundary by one frame:
85      segmenters.add(new SegmenterShift("\\segShift", c));
86      // ------------------------------------------------------------------
87      // shift every reference boundary by 10% of the segment length
88      segmenters.add(new SegmenterRelativeBias("\\segTen", c, 0.1));
89      // ------------------------------------------------------------------
90      // shift every reference boundary by 5% of the segment length
91      segmenters.add(new SegmenterRelativeBias("\\segFive", c, 0.05));
92
93      this.runSegmenterComparison(
94            c,
95            segmenters.toArray(new ISegmenter[segmenters.size()]),
96            toleranceInMs,
97            1,
98            "resultDiagnostic",
```

```
99            2, 6 );
100      // Using 2 decimal digits for the LaTeX tables and 6 for the csv files
101      // for readability and higher precision in subsequent analysis,
102      // respectively. (same for baseline segmenters)
103   }
104
105   /**
106    * Wrapper to start the comparison with the
107    * constant baseline segmentation.
108    * @param c
109    * @param toleranceInMs
110    * @param alpha
111    */
112   public void runBaselineConstSegmenter(
113         CorpusReader c,
114         double toleranceInMs,
115         double alpha )
116   {
117      this.runSegmenterComparison(
118            c, new ISegmenter[]{new BaselineConst("\\segConst", c, alpha)},
119            toleranceInMs, 1, "resultBaselineConst", 2, 6 );
120   }
121
122   /**
123    * Wrapper to start the comparison with the
124    * random baseline segmentation.
125    * @param c
126    * @param toleranceInMs
127    * @param alpha
128    * @param runs
129    */
130   public void runBaselineRandSegmenter(
131         CorpusReader c,
132         double toleranceInMs,
133         double alpha,
134         int runs )
135   {
136      ISegmenter[] segmenters = new ISegmenter[]{ new BaselineRandNorm("\\segRand",↩
          c, alpha, 2) };
137      this.runSegmenterComparison(c, segmenters, toleranceInMs, runs, "↩
         resultBaselineRandAverage", 2, 6);
138   }
139
140   /**
141    * @param c -- the input reference data
142    * @param segmeters -- the segmenters to be applied to the input data
143    * @param toleranceInMs -- the size of the tolerance
144    * @param runs -- the number of runs that each segmenter should be applied
145    *                to the input data
```

```
146     * @param outputFileName -- the file name and path of the output
147     * @param resDigitsTeX -- the decimal digits to be printed in the TeX output
148     * @param resDigitsTab -- the decimal digits to be printed in the CSV output
149     */
150    public void runSegmenterComparison(
151          CorpusReader c,
152          ISegmenter[] segmeters,
153          //Evaluator eval,
154          double toleranceInMs,
155          int runs,
156          String outputFileName,
157          int resDigitsTeX,
158          int resDigitsTab)
159    {
160       File outputFileTab = new File(outDir, outputFileName+".tab");
161       File outputFileTex = new File(outDir, outputFileName+".tex");
162       Evaluator eval = new Evaluator(toleranceInMs, c.getFrameRate(), true, true);
163       String resFormatTeX = "%,."+resDigitsTeX+"f";
164       String resFormatTab = "%."+resDigitsTab+"f";
165       TabFileWriter outTab = new TabFileWriter(outputFileTab, "\t");
166       TabFileWriter outTex = new TabFileWriter(outputFileTex, "\t&␣", "\\\\");
167       String[] head = new String[]{"␣", "matched", "TP", "prec", "rec", "F1", "R", ↩
             "OS", "S'", "Pk", "WD", "relTP", "relPrec", "relRec", "relF1"};
168       // write header to result tab file
169       outTab.writeLine(head);
170       // write header to result tex file
171       outTex.writeLine(new String[]{"%␣" + outputFileTex.getAbsolutePath()});
172       outTex.writeLine(head);
173       // print tolerance window statistics:
174       try {
175          for(int i=0; i < segmeters.length; i++){
176             ISegmenter seg = segmeters[i];
177             System.out.println("evaluating:␣" + seg.getTag());
178             EvaluationResults resTotal;
179             if(runs==1){
180                int[] hyp = seg.getBoundaryIndices();
181                resTotal = eval.evaluateSegmentation(c, hyp, seg.getTag());
182             } else {
183                EvaluationResults[] allResults = new EvaluationResults[runs];
184                for(int r=0; r<runs; r++){
185                   System.out.print(".");
186                   int[] hyp = seg.getBoundaryIndices();
187                   EvaluationResults resLocal = eval.evaluateSegmentation(c, hyp, ↩
                         seg.getTag());
188                   allResults[r] = resLocal;
189                   if(r!=0 && r%100==0){
190                      System.out.println();
191                   }
192                }
```

```
193                 System.out.println();
194                 resTotal = this.getAverageResult(allResults);
195             }
196             this.appendResult(outTab, resFormatTab, resTotal, toleranceInMs, c.↩
                   getFrameRate(), false);
197             this.appendResult(outTex, resFormatTeX, resTotal, toleranceInMs, c.↩
                   getFrameRate(), true);
198          }
199       } catch (Exception e) {
200          e.printStackTrace();
201       }
202       if( outTab.close() ){
203          System.out.println("Results␣written␣to␣" + outputFileTab.getAbsolutePath()↩
                );
204       }
205       if( outTex.close() ){
206          System.out.println("␣␣␣␣␣␣␣␣␣␣␣␣␣and␣" + outputFileTex.getAbsolutePath()↩
                );
207       }
208    }
209
210    private EvaluationResults getAverageResult(EvaluationResults[] allResults)
211    {
212       EvaluationResults avg;
213       if( allResults.length == 1 ){
214          avg=allResults[0];
215       } else {
216          avg=null;
217          String tag      = allResults[0].tag;
218          int totalLength = allResults[0].totalFrames;
219          int refLen = allResults[0].referenceLength;
220          int hypLen = allResults[0].hypothesisLength;
221          int tolerance = allResults[0].tolerance;
222          avg = new EvaluationResults(tag, totalLength, refLen, hypLen, tolerance);
223
224          double sumMatchedBoundaries = allResults[0].matchedBoundaries;
225          double sumTruePositives     = allResults[0].truePositives;
226          double sumFalsePositives    = allResults[0].falsePositives;
227          double sumFalseNegatives    = allResults[0].falseNegatives;
228          double sumPk                = allResults[0].pk;
229          double sumWindowDiff        = allResults[0].windowDiff;
230          double sumPrecision         = allResults[0].precision;
231          double sumRecall            = allResults[0].recall;
232          double sumF1                = allResults[0].F1;
233          double sumOS                = allResults[0].oversegmentation;
234          double sumSER               = allResults[0].SERbar;
235          double sumR                 = allResults[0].R;
236          double sumRelTP             = allResults[0].relativeTP;
237          double sumRelPrecision      = allResults[0].relativePrecision;
```

```
238         double sumRelRecall         = allResults[0].relativeRecall;
239         double sumRelF              = allResults[0].relativeF1;
240
241         for(int i=1; i<allResults.length; i++){
242            // add to sum
243            sumMatchedBoundaries += allResults[i].matchedBoundaries;
244            sumTruePositives     += allResults[i].truePositives;
245            sumFalsePositives    += allResults[i].falsePositives;
246            sumFalseNegatives    += allResults[i].falseNegatives;
247            sumPk                += allResults[i].pk;
248            sumWindowDiff        += allResults[i].windowDiff;
249            sumPrecision         += allResults[i].precision;
250            sumRecall            += allResults[i].recall;
251            sumF1                += allResults[i].F1;
252            sumOS                += allResults[i].oversegmentation;
253            sumSER               += allResults[i].SERbar;
254            sumR                 += allResults[i].R;
255            sumRelTP             += allResults[i].relativeTP;
256            sumRelPrecision      += allResults[i].relativePrecision;
257            sumRelRecall         += allResults[i].relativeRecall;
258            sumRelF              += allResults[i].relativeF1;
259         }
260
261         double runs = allResults.length;
262         avg.matchedBoundaries = (int)Math.round(sumMatchedBoundaries / runs);
263         avg.truePositives     = (int)Math.round(sumTruePositives / runs);
264         avg.falsePositives    = (int)Math.round(sumFalsePositives / runs);
265         avg.falseNegatives    = (int)Math.round(sumFalseNegatives / runs);
266         avg.pk                = sumPk / runs;
267         avg.windowDiff        = sumWindowDiff / runs;
268         avg.precision         = sumPrecision / runs;
269         avg.recall            = sumRecall / runs;
270         avg.F1                = sumF1 / runs;
271         avg.oversegmentation  = sumOS / runs;
272         avg.SERbar            = sumSER / runs;
273         avg.R                 = sumR / runs;
274         avg.relativeTP        = sumRelTP / runs;
275         avg.relativePrecision = sumRelPrecision / runs;
276         avg.relativeRecall    = sumRelRecall / runs;
277         avg.relativeF1        = sumRelF / runs;
278      }
279      return avg;
280   }
281
282   private void appendResult(
283         TabFileWriter out,
284         String resFormat,
285         EvaluationResults res,
286         double toleranceInMs,
```

```
287         int frameRate,
288         boolean printToOut )
289   {
290      String[] rline = new String[]{
291            res.tag,
292            String.format(this.loc,"%,d",res.matchedBoundaries),
293            String.format(this.loc,"%,d",res.truePositives),
294            String.format(this.loc,resFormat,res.precision),
295            String.format(this.loc,resFormat,res.recall),
296            String.format(this.loc,resFormat,res.F1),
297            String.format(this.loc,resFormat,res.R),
298            String.format(this.loc,resFormat,res.oversegmentation),
299            String.format(this.loc,resFormat,res.SERbar),
300            String.format(this.loc,resFormat,res.pk),
301            String.format(this.loc,resFormat,res.windowDiff),
302            String.format(this.loc,resFormat,res.relativeTP),
303            String.format(this.loc,resFormat,res.relativePrecision),
304            String.format(this.loc,resFormat,res.relativeRecall),
305            String.format(this.loc,resFormat,res.relativeF1),
306      };
307      out.writeLine(rline);
308
309      if(printToOut){
310         StringBuilder resultsText = new StringBuilder();
311         resultsText.append("----------\n["+res.tag+"]\n");
312         resultsText.append("Total␣size:␣" + String.format(this.loc,"%,d",res.↩
               totalFrames) + "␣frames.\n");
313         resultsText.append("Boundaries␣in␣test␣␣␣␣␣=␣" + String.format(this.loc,"↩
               %,d", res.hypothesisLength) );
314         resultsText.append("\n");
315         resultsText.append("Boundaries␣in␣reference␣=␣" + String.format(this.loc,"↩
               %,d", res.referenceLength) );
316         resultsText.append("\n");
317         resultsText.append("Tolerance␣window␣=␣+/-␣"+ String.format(this.loc,"↩
               %,4.2f",toleranceInMs) );
318         resultsText.append("ms␣=␣+/-␣"+ res.tolerance + "␣frames.\n\n");
319         resultsText.append("matched␣␣␣␣␣␣␣=␣" + String.format(this.loc,"%,9d", ↩
               res.matchedBoundaries));
320         resultsText.append("\n");
321         resultsText.append("true␣positive␣␣=␣" + String.format(this.loc,"%,9d", ↩
               res.truePositives));
322         resultsText.append("\n\n");
323         resultsText.append("precision␣␣=␣" + String.format(this.loc,"%7.4f", res.↩
               precision) );
324         resultsText.append("\n");
325         resultsText.append("recall␣␣␣␣=␣" + String.format(this.loc,"%7.4f", res.↩
               recall) );
326         resultsText.append("\n");
```

```
327         resultsText.append("F-measure␣␣=␣" + String.format(this.loc,"%7.4f",res.F1↩
               ) );
328         resultsText.append("\n\n");
329         // relative TP and F1
330         resultsText.append("relative␣true␣positives␣␣=␣" + String.format(this.loc,↩
               "%14.4f", res.relativeTP));
331         resultsText.append("\n");
332         resultsText.append("relative␣precision␣␣=␣" + String.format(this.loc,"%7.4↩
               f", res.relativePrecision) );
333         resultsText.append("\n");
334         resultsText.append("relative␣recall␣␣␣␣=␣" + String.format(this.loc,"%7.4↩
               f", res.relativeRecall) );
335         resultsText.append("\n");
336         resultsText.append("relative␣F-measure␣␣=␣" + String.format(this.loc,"%7.4↩
               f",res.relativeF1) );
337         resultsText.append("\n\n");
338         // R-value
339         resultsText.append("R-value␣␣␣␣=␣"+ String.format(this.loc,"%7.4f", res.R)↩
               );
340         resultsText.append("\n");
341         resultsText.append("OS␣␣␣␣␣␣␣␣=␣" + String.format(this.loc,"%.4f", res.↩
               oversegmentation) + "%");
342         if(res.oversegmentation<0.0){
343            resultsText.append("␣(under-segmentation)");
344         }
345         resultsText.append("\n");
346         resultsText.append("SER'␣␣␣␣␣␣=␣" + String.format(this.loc,"%.4f", res.↩
               SERbar));
347         resultsText.append("\n");
348         // WindowDiff
349         resultsText.append("P_k␣␣␣␣␣␣␣=␣" + String.format(this.loc,"%.4f", res.pk↩
               ));
350         resultsText.append("\n");
351         resultsText.append("WindowDiff␣=␣" + String.format(this.loc,"%.4f", res.↩
               windowDiff));
352         resultsText.append("\n");
353         resultsText.append("␣(k=" + res.k + "␣frames;␣"+(res.k*1000/frameRate)+"ms↩
               )\n");
354         System.out.println(resultsText.toString());
355      }
356   }
357
358   public void printToleranceWindowStatistics(CorpusReader c, double toleranceInMs)
359   {
360      int t  = (int) Math.ceil( toleranceInMs / 1000.0 * c.getFrameRate() );
361      int t2 = t+t;
362      int[] bnd = c.getBoundaryIndices();
363      int coverage = 0;
364      int overlappingTolerances = 0;
```

```
365         int tightTolerances = 0;
366         // first segment: (from start of sequence to the first boundary)
367         // there cannot be overlap in the very fist segment, since there is no
368         // tolerance around the absolute start point of the sequence.
369         int segmentLength = bnd[0];
370         if( segmentLength < t ){
371            coverage += segmentLength;
372         } else {
373            coverage += t;
374         }
375         // internal segments:
376         int lastStart = bnd[0];
377         for(int b=1; b<bnd.length; b++){
378            segmentLength = bnd[b] - lastStart + 1;
379
380            if( segmentLength > t2 ){
381               // no overlap
382               coverage += t2;
383            } else if (segmentLength < t2){
384               // overlapping tolerance windows
385               coverage += segmentLength;
386               overlappingTolerances++;
387            } else {
388               // tight fit
389               coverage += segmentLength;
390               tightTolerances++;
391            }
392            lastStart = bnd[b];
393         }
394         // last segment: (from last boundary to the end of the sequence)
395         // as for the very first segment, there cannot be any overlap in the
396         // very last segment, since there is no tolerance around the absolute
397         // end of the sequence.
398         segmentLength = c.getNumberOfFrames() - bnd[bnd.length-1] ;
399         if( segmentLength < t ){
400            coverage += segmentLength;
401         } else {
402            coverage += t;
403         }
404         double covratio = (double)coverage / (double)c.getNumberOfFrames() * 100.0;
405         System.out.println("Tolerance␣window␣statistics:");
406         System.out.println("␣window␣size:␣+/-␣"
407               + String.format(this.loc,"%,4.2f",toleranceInMs)
408               + "␣ms␣==␣+/-␣"+ t + "␣frames." );
409         System.out.println("␣overlapping␣tolerance␣windows:␣" + String.format(this.↩
            loc,"%,d",overlappingTolerances));
410         System.out.println("␣␣␣␣␣␣␣tight␣tolerance␣windows:␣" + String.format(this.↩
            loc,"%,d",tightTolerances));
```

```
411        System.out.println("␣␣␣␣␣␣totally␣covered␣segments:␣" + String.format(this.↩
              loc,"%,d",(overlappingTolerances+tightTolerances)));
412        System.out.println("␣tolerance␣\"coverage\":␣"
413              + String.format(this.loc,"%,d",coverage) + "/" + String.format(this.loc↩
                 ,"%,d",c.getNumberOfFrames()) + "␣frames" );
414        System.out.println("␣tolerance␣\"coverage\":␣"
415              + String.format(this.loc,"%,.4f",covratio) + "%␣of␣the␣corpus'␣frames" ↩
                 );
416     }
417  }
```

Listing B.7: ISegmenter.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  /**  All segmenters must implement this interface */
4  public interface ISegmenter {
5
6     public double[] getBoundariness();
7
8     public int[] getBoundaryIndices();
9
10    /**
11     * @return a unique tag for this segmenter (for logging)
12     */
13    public String getTag();
14 }
```

Listing B.8: SegmenterPerfect.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import de.uniStuttgart.danielDuran.ICorpusReader;
4
5  public class SegmenterPerfect implements ISegmenter {
6
7     private final String tag;
8     private final int[] ref;
9
10    public SegmenterPerfect(String tag, ICorpusReader c) {
11       this.tag = tag;
12       this.ref = c.getBoundaryIndices();
13    }
14
15    public double[] getBoundariness() {
16       System.err.println("getBoundariness␣not␣implemented␣in␣SegmenterPerfect,␣↩
            returning␣NULL!");
17       return null;
18    }
```

```
19
20    public int[] getBoundaryIndices() {
21       return ref;
22    }
23
24    public String getTag() {
25       return this.tag;
26    }
27 }
```

Listing B.9: SegmenterTotal.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import de.uniStuttgart.danielDuran.ICorpusReader;
4
5  public class SegmenterTotal implements ISegmenter {
6
7     private final String tag;
8     private final int[] indices;
9
10    public SegmenterTotal(String tag, ICorpusReader c) {
11       this.tag = tag;
12       int len = c.getNumberOfFrames();
13       this.indices = new int[len-1];
14       for(int i=0; i<len-1; i++ ){
15          this.indices[i] = i+1;
16       }
17    }
18
19    public double[] getBoundariness() { return null; }// not used
20
21    public int[] getBoundaryIndices() {
22       return this.indices;
23    }
24
25    public String getTag() {
26       return this.tag;
27    }
28 }
```

Listing B.10: SegmenterCenter.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import de.uniStuttgart.danielDuran.ICorpusReader;
4
5  public class SegmenterCenter implements ISegmenter {
6
```

```
7     private final String tag;
8     private final int[] indices;
9
10    public SegmenterCenter(String tag, ICorpusReader c) {
11       this.tag=tag;
12       int[] ref = c.getBoundaryIndices();
13       this.indices = new int[ref.length + 1];
14       double lastR=0.0;
15       int lastHyp = -1;
16       int ptr=0;
17       for(int i=0; i<ref.length; i++ ){
18          double r = ref[i];
19          // determine the center of the previous segment
20          int z = (int)(lastR + Math.round(  (r - lastR) / 2.0  ));
21          if(z <= lastHyp){
22             // avoid zero-length segments
23             System.out.println("␣***␣WARN␣***␣" +
24                   "Shifting␣hyp␣boundary␣in␣" + this.tag
25                   + "␣by␣one␣frame␣@␣" + z);
26             z = z+1;
27          }
28          this.indices[ptr++] = z;
29          lastHyp=z;
30          lastR=r;
31       }
32       // add boundary at center of last segment
33       this.indices[ptr++] = (int)(lastR + Math.round(  (c.getNumberOfFrames() - ↩
            lastR) / 2.0 ));
34    }
35
36    public double[] getBoundariness() { return null; }// not used
37
38    public int[] getBoundaryIndices() {
39       return this.indices;
40    }
41
42    public String getTag() {
43       return this.tag;
44    }
45 }
```

Listing B.11: SegmenterDeltaMinusOne.java

```
1 package de.uniStuttgart.danielDuran.segmenter;
2
3 import de.uniStuttgart.danielDuran.ICorpusReader;
4
5 public class SegmenterDeltaMinusOne implements ISegmenter {
6
```

```
7     private final String tag;
8     private final int[] indices;
9
10    public SegmenterDeltaMinusOne(
11          String tag,
12          ICorpusReader c,
13          double toleranceMs )
14    {
15       this.tag = tag;
16       int f = c.getFrameRate();
17       int deltaMinus1 = (int) Math.round( toleranceMs / 1000.0 * f ) - 1;
18       int[] ref = c.getBoundaryIndices();
19       int len   = ref.length;
20       // check whether the last boundary can be shifted by delta+1 frames
21       if(  ref[ref.length-1] + deltaMinus1 >= c.getNumberOfFrames() ){
22          len = len -1;
23       }
24       this.indices = new int[len];
25       for(int i=0; i<len; i++){
26          this.indices[i] = ref[i] + deltaMinus1;
27       }
28    }
29
30    public double[] getBoundariness() { return null; }// not used
31
32    public int[] getBoundaryIndices() {
33       return this.indices;
34    }
35
36    public String getTag() {
37       return this.tag;
38    }
39 }
```

Listing B.12: SegmenterDeltaPlusOne.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import de.uniStuttgart.danielDuran.ICorpusReader;
4
5  public class SegmenterDeltaPlusOne implements ISegmenter {
6
7     private final String tag;
8     private final int[] indices;
9
10    public SegmenterDeltaPlusOne(
11          String tag,
12          ICorpusReader c,
13          double toleranceMs )
```

```
14    {
15       this.tag = tag;
16       int f    = c.getFrameRate();
17       int deltaPlus1 = (int) Math.round( toleranceMs / 1000.0 * f ) + 1;
18       int[] ref = c.getBoundaryIndices();
19       int len   = ref.length;
20       // check whether the last boundary can be shifted by delta+1 frames
21       if(  ref[ref.length-1] + deltaPlus1 >= c.getNumberOfFrames() ){
22          len = len -1;
23       }
24       this.indices = new int[len];
25       for(int i=0; i<len; i++) {
26          this.indices[i] = ref[i] + deltaPlus1;
27       }
28    }
29
30    public double[] getBoundariness() { return null; }// not used
31
32    public int[] getBoundaryIndices() {
33       return this.indices;
34    }
35
36    public String getTag() {
37       return this.tag;
38    }
39 }
```

Listing B.13: SegmenterDouble.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import java.util.Arrays;
4
5  import de.uniStuttgart.danielDuran.ICorpusReader;
6
7  public class SegmenterDouble implements ISegmenter {
8
9     private final String tag;
10    private final int[] indices;
11
12    public SegmenterDouble(String tag, ICorpusReader c) {
13       this.tag=tag;
14       int[] ref  = c.getBoundaryIndices();
15       int hypLen = ref.length * 2;
16       int[] hypIndices = new int[hypLen];
17       int hidx = 0;
18       int lastHyp = -1;
19       for(int i=0; i<ref.length; i++) {
20          // If two consecutive reference boundaries are only 1 frame apart,
```

```
21          // a simple replacement would generate a hyp sequence which is not
22          // monotonically increasing. Therefore, we need to check, whether
23          // a generated boundary is larger then the previously generated one
24          int a = ref[i] - 1;
25          int b = ref[i] + 1;
26          if( a > lastHyp){
27             hypIndices[hidx++] = a;
28             lastHyp = a;
29          }
30          if(b > lastHyp){
31             hypIndices[hidx++] = b;
32             lastHyp = b;
33          }
34       }
35       if(hidx==hypLen){
36          this.indices = hypIndices;
37       } else {
38          this.indices = Arrays.copyOf(hypIndices, hidx);
39          System.out.println("␣***␣WARN␣***␣Less␣boundaries␣in␣" + this.tag
40                + "␣than␣expected:␣ref*2="+ hypLen + ";␣hyp="+hidx
41          );
42       }
43    }
44
45    public double[] getBoundariness() { return null; }// not used
46
47    public int[] getBoundaryIndices() {
48       return this.indices;
49    }
50
51    public String getTag() {
52       return this.tag;
53    }
54 }
```

Listing B.14: SegmenterHalfRef.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import de.uniStuttgart.danielDuran.ICorpusReader;
4
5  public class SegmenterHalfRef implements ISegmenter {
6
7     private final String tag;
8     private final int[] indices;
9
10    public SegmenterHalfRef(String tag, ICorpusReader c) {
11       this.tag  = tag;
12       int[] ref = c.getBoundaryIndices();
```

```
13       int len = (int)Math.round((double)ref.length / 2.0);
14       this.indices = new int[len];
15       int j=0;
16       boolean sw=true;
17       for(int i=0; i<ref.length; i++){
18          if(sw){
19             this.indices[j] = ref[i];
20             j++;
21             sw=false;
22          } else {
23             sw=true;
24          }
25       }
26    }
27
28    public double[] getBoundariness() { return null; }// not used
29
30    public int[] getBoundaryIndices() {
31       return this.indices;
32    }
33
34    public String getTag() {
35       return this.tag;
36    }
37 }
```

Listing B.15: SegmenterHalfSegments.java

```
1  package de.uniStuttgart.danielDuran.segmenter;
2
3  import java.util.Arrays;
4
5  import de.uniStuttgart.danielDuran.ICorpusReader;
6
7  public class SegmenterHalfSegments implements ISegmenter {
8
9     private final String tag;
10    private final int[] indices;
11
12    public SegmenterHalfSegments(String tag, ICorpusReader c) {
13       this.tag=tag;
14       int[] ref = c.getBoundaryIndices();
15       int len = (ref.length *2) + 1;
16       int[] hypIndices = new int[len];
17       double lastR=0.0;
18       int lastHyp = -1;
19       int ptr=0;
20       for(int i=0; i<ref.length; i++ ) {
21          double r = ref[i];
```

```
22          // determine the center of the previous segment
23          int z = (int)(lastR + Math.round(  (r - lastR) / 2.0  ));
24          if( z <= lastHyp){
25             // avoid zero-length segments
26             System.out.println("␣***␣WARN␣***␣Shifting␣hyp␣boundary␣in␣"
27                   + this.tag  + "␣by␣one␣frame␣@␣" + z);
28             z = z+1;
29          }
30          hypIndices[ptr++] = z;
31          lastHyp = z;
32          if(ref[i] <= lastHyp){
33             System.out.println("␣***␣WARN␣***␣Dropping␣ref␣boundary␣in␣"
34                   + this.tag +"␣@␣"+ref[i]+"␣to␣avoid␣zero-segment.");
35          } else {
36             hypIndices[ptr++] = ref[i];
37             lastHyp = ref[i];
38          }
39          lastR=r;
40       }
41       // add boundary at center of last segment
42       hypIndices[ptr++] = (int)(lastR + Math.round(  (c.getNumberOfFrames() - lastR↩
            ) / 2.0 ));
43
44       if(ptr==len){
45          this.indices = hypIndices;
46       } else {
47          this.indices = Arrays.copyOf(hypIndices, ptr);
48          System.out.println("␣***␣WARN␣***␣Less␣boundaries␣in␣" + this.tag
49                + "␣than␣expected:␣ref*2+1="+ len + ";␣hyp="+ptr
50                );
51       }
52    }
53
54    public double[] getBoundariness() { return null; }// not used
55
56    public int[] getBoundaryIndices() {
57       return this.indices;
58    }
59
60    public String getTag() {
61       return this.tag;
62    }
63 }
```

Listing B.16: SegmenterNo.java

```
1 package de.uniStuttgart.danielDuran.segmenter;
2
3 /**
```

```
 * A segmenter which produces no boundaries
 */
public class SegmenterNo implements ISegmenter {

   private final String tag;

   public SegmenterNo(String tag) {
      this.tag=tag;
   }

   public double[] getBoundariness() { return null; }// not used

   public int[] getBoundaryIndices() {
      return new int[0];
   }

   public String getTag() {
      return this.tag;
   }
}
```

Listing B.17: SegmenterShift.java

```
package de.uniStuttgart.danielDuran.segmenter;

import de.uniStuttgart.danielDuran.ICorpusReader;

public class SegmenterShift implements ISegmenter {

   private final String tag;
   private final int[] indices;

   public SegmenterShift(String tag, ICorpusReader c) {
      this.tag=tag;
      int[] ref = c.getBoundaryIndices();
      this.indices = new int[ref.length];
      int lastHyp = -1;
      for(int i=0; i<ref.length; i++){
         int h =  ref[i] + 1;
         if(h <= lastHyp){
            // avoid zero-length segments: instead of simply dropping this
            // boundary, we shift it in order to preserve the total number
            // of boundaries. ATTENTION: this might fail if there is a long
            // sequence of adjacent boundaries!
            System.out.println("␣***␣WARN␣***␣Shifting␣hyp␣boundary␣in␣"
                  + this.tag + "␣by␣one␣frame␣@␣" + h);
            h = h+1;
         }
         this.indices[i] = h;
```

```
27          lastHyp = h;
28       }
29    }
30
31    public double[] getBoundariness() { return null; }// not used
32
33    public int[] getBoundaryIndices() {
34       return this.indices;
35    }
36
37    public String getTag() {
38       return this.tag;
39    }
40 }
```

Listing B.18: BaselineConst.java

```
1 package de.uniStuttgart.danielDuran.segmenter;
2
3 import de.uniStuttgart.danielDuran.ICorpusReader;
4
5 public class BaselineConst implements ISegmenter {
6
7    private final String tag;
8    private final int[] indices;
9
10   public BaselineConst(String tag, ICorpusReader c, double alpha) {
11      int[] ref = c.getBoundaryIndices();
12      int targetMean = (int)Math.round( (double)c.getNumberOfFrames() / (c.↩
           getNumberOfSegments() * alpha) );
13      int len = (int)Math.round( ref.length * alpha);
14      this.indices = new int[len];
15      int idx=targetMean;
16      for(int i=0; i<len; i++){
17         this.indices[i] = idx;
18         idx = idx + targetMean;
19      }
20      this.tag=tag;
21   }
22
23   public double[] getBoundariness() { return null; }// not used
24
25   public int[] getBoundaryIndices() {
26      return this.indices;
27   }
28
29   public String getTag() {
30      return this.tag;
31   }
```

```
}
```

Listing B.19: BaselineRandNorm.java

```
package de.uniStuttgart.danielDuran.segmenter;

import java.util.Random;

import de.uniStuttgart.danielDuran.ICorpusReader;
import de.uniStuttgart.danielDuran.tools.SeqStat;

public class BaselineRandNorm implements ISegmenter {

   private final Random generator;
   private final String tag;
   private final int[] indices;

   /**
    * @param tag   -- a label for the segmenter which will be used
    *                 in the output table
    * @param c     -- the input data
    * @param alpha -- the parameter defining the total number of generated
    *                 segments. A value of alpha &lt; 1 results in
    *                 undersegmentation, and a value of alpha &gt; 1 results
    *                 in oversegmentation.
    * @param maxSDfactor -- the factor controlling the maximum
    *                       length of the generated segments, which is
    *    <code>MIN(true max, true mean + maxSDfactor * sd)</code>
    */
   public BaselineRandNorm(
         String tag,
         ICorpusReader c,
         double alpha,
         int maxSDfactor )
   {
      this.generator = new Random();
      this.tag=tag;

      SeqStat refLengths = c.getSegmentLengthStats();

      double trueMean   = refLengths.getMean();
      double trueSD     = refLengths.getSD();
      double scaledMean = trueMean / alpha;
      double scaledSD   = trueSD   / alpha;
      double scaledMin  = refLengths.getMean() / alpha;
      double scaledMax  = refLengths.getMax() / alpha;

      if( scaledMax > (scaledMean + (maxSDfactor*scaledSD))) {
         System.out.print("[BaselineRandNorm]␣Re-setting␣max␣from␣" + scaledMax);
```

```
46          scaledMax = scaledMean + (maxSDfactor*scaledSD);
47          System.out.print("␣to␣" + scaledMax + "\n");
48       }
49
50       int scMin = (int)Math.round(scaledMin);
51       int scMax = (int)Math.round(scaledMax);
52       int[] ref = c.getBoundaryIndices();
53       int len = (int)Math.round( ref.length * alpha);
54       this.indices = new int[len];
55       int idx=0;
56       for(int i=0; i<len; i++)
57       {
58          int nextLength = this.getNextGaussianInt(scaledMean, scaledSD);
59          if(nextLength<scMin){
60             nextLength = scMin;
61          } else if(nextLength>scMax){
62             nextLength = scMax;
63          }
64          idx = idx + nextLength;
65          this.indices[i] = idx;
66       }
67    }
68
69    private int getNextGaussianInt(double mean, double sd) {
70       double r = this.generator.nextGaussian();
71       return (int)Math.round((r * sd) + mean);
72    }
73
74    public double[] getBoundariness() { return null; }// not used
75
76    public int[] getBoundaryIndices() {
77       return this.indices;
78    }
79
80    public String getTag() {
81       return this.tag;
82    }
83 }
```

Listing B.20: TabFileWriter.java

```
1 package de.uniStuttgart.danielDuran.tools;
2
3 import java.io.BufferedOutputStream;
4 import java.io.File;
5 import java.io.FileOutputStream;
6 import java.io.IOException;
7 import java.io.OutputStreamWriter;
8
```

```
9  public class TabFileWriter {
10
11    private final String colSep;
12    private final String lineEnd;
13    private OutputStreamWriter out;
14
15    public TabFileWriter(File outputFile, String colSep) {
16       this(outputFile, colSep, "");
17    }
18
19    public TabFileWriter(File outputFile, String colSep, String lineEnd) {
20       try {
21          out  = new OutputStreamWriter(new BufferedOutputStream(new ↩
               FileOutputStream(outputFile)), "UTF-8");
22       } catch (Exception e) {
23          e.printStackTrace();
24       }
25       this.colSep = colSep;
26       this.lineEnd = lineEnd+"\n";
27    }
28
29    public void writeLine(String[] line) {
30       if(null!=this.out && null!=line){
31          StringBuilder sb = new StringBuilder();
32          sb.append(line[0]);
33          for(int i=1; i<line.length; i++){
34             sb.append(this.colSep);
35             sb.append(line[i]);
36          }
37          sb.append(this.lineEnd);
38          try {
39             this.out.write(sb.toString());
40             this.out.flush();
41          } catch (IOException e) {
42             e.printStackTrace();
43          }
44       }
45    }
46
47    public boolean close() {
48       boolean ok = true;
49       if(null!=this.out){
50          try {
51             this.out.flush();
52             this.out.close();
53             this.out = null;
54          } catch (IOException e) {
55             e.printStackTrace();
56             ok = false;
```

```
57          }
58       }
59       return ok;
60    }
61 }
```

Listing B.21: Evaluator.java

```
1 package de.uniStuttgart.danielDuran.evaluation;
2
3 import de.uniStuttgart.danielDuran.CorpusReader;
4 import de.uniStuttgart.danielDuran.tools.SeqStat;
5
6 /**
7  * @author Daniel Duran
8  */
9 public class Evaluator {
10
11    /** tolerance in frames */
12    private final int tolerance;
13    private final boolean computeWindowDiff;
14    private final boolean computePk;
15
16    public Evaluator(
17          double toleranceInMs,
18          int frameRate,
19          boolean computePk,
20          boolean computeWindowDiff )
21    {
22       this.tolerance = (int) Math.ceil( toleranceInMs / 1000.0 * frameRate );
23       this.computePk = computePk;
24       this.computeWindowDiff = computeWindowDiff;
25    }
26
27    public EvaluationResults evaluateSegmentation(
28          CorpusReader c,
29          int[] hyp,
30          String tag )
31    {
32       int[] ref = c.getBoundaryIndices();
33       SeqStat refLengths = c.getSegmentLengthStats();
34       int totalLength = c.getNumberOfFrames();
35       EvaluationResults res = new EvaluationResults(tag, totalLength, ref.length, ↩
            hyp.length, this.tolerance);
36       // align the two sequences of boundaries:
37       this.align(ref, hyp, totalLength, res);
38
39       if(this.computePk || this.computeWindowDiff){
40          res.k = (int)Math.round( refLengths.getMean() / 2.0);
```

```
41          int pkCount=0;
42          int wdCount=0;
43          int ptrRef=0;
44          int ptrHyp=0;
45          int refSegEnd=0;// the end of the current segment in ref
46          int hypSegEnd=0;// the end of the current segment in hyp
47          int end = totalLength - res.k;
48          for(int i=0; i < end; i++){
49             int probeEnd = i+res.k;
50             if(i>refSegEnd){
51                ptrRef++;
52                if(ptrRef<ref.length){
53                   refSegEnd = ref[ptrRef];
54                } else {
55                   refSegEnd = totalLength;
56                }
57             }
58             if(i>hypSegEnd){
59                ptrHyp++;
60                if(ptrHyp<hyp.length){
61                   hypSegEnd = hyp[ptrHyp];
62                } else {
63                   hypSegEnd = totalLength;
64                }
65             }
66             if(this.computePk){
67                boolean sameRef = true;
68                if( ptrRef < ref.length ){
69                   sameRef = probeEnd <= ref[ptrRef];
70                }
71                // else: the last segment has been reached, so
72                // all probes are within the same segment
73                boolean sameHyp = true;
74                if( ptrHyp < hyp.length ){
75                   sameHyp = probeEnd <= hyp[ptrHyp];
76                }
77                if( sameRef != sameHyp ){
78                   pkCount++;
79                }
80             }
81             if(this.computeWindowDiff){
82                int bRef = this.getNumberOfBoundaries(i, probeEnd, ref, ptrRef);
83                int bHyp = this.getNumberOfBoundaries(i, probeEnd, hyp, ptrHyp);
84                if(bRef != bHyp){
85                   wdCount++;
86                }
87             }
88          }//END:for
89          res.pk = (double)pkCount / ((double)totalLength - (double)res.k);
```

```
90          res.windowDiff = (1.0 / ((double)totalLength - (double)res.k)) * (double)↩
               wdCount;
91       }//END:if Pk|WindowDiff
92       // ------------------------------------
93       // compute results based on TP
94       res.falseNegatives = ref.length - res.truePositives;
95       res.falsePositives = hyp.length - res.truePositives;
96       double H  = (double)hyp.length;
97       double R  = (double)ref.length;
98       double TP = (double)res.truePositives;
99       res.precision = TP / H;
100      res.recall    = TP / R;
101      res.F1 = Fmeasure.getF1(res.precision, res.recall);
102      res.relativePrecision = res.relativeTP / H;
103      res.relativeRecall    = res.relativeTP / R;
104      res.relativeF1        = Fmeasure.getF1(res.relativePrecision, res.↩
            relativeRecall);
105      res.oversegmentation = ((H / R) - 1.0) * 100.0;
106      res.SERbar = (double)(res.falseNegatives + res.falsePositives) / R;
107      double t1 = 100.0 - (res.recall*100.0);
108      double r1 = Math.sqrt( (t1*t1) + (res.oversegmentation * res.oversegmentation↩
            ) );
109      double r2 =( -res.oversegmentation + (res.recall*100.0) - 100.0 ) / Math.sqrt↩
            (2.0);
110      res.R = 1.0 - ( (Math.abs(r1) + Math.abs(r2)) / 200.0 );
111      return res;
112   }
113
114   private int getNumberOfBoundaries(int startIndex, int endIndex, int[] seq, int ↩
         ptrSeq) {
115      int n = 0;
116      while(ptrSeq < seq.length && seq[ptrSeq] < endIndex){
117         n++;
118         ptrSeq++;
119      }
120      return n;
121   }
122
123   private void align(int[] ref, int[] hyp, int totalLength, EvaluationResults res)↩
          {
124      double leftMargin = ref[0] / 2.0;
125      double rightMargin = ref[ref.length-1] + (totalLength - ref[ref.length-1]) / ↩
            2.0;
126      int lastMatchedHyp = 0;
127      int lastMatchedRef = 0;
128      /* these "pointers" are only needed to speed up the search!
129       * the results are the same, when calling
130       *     this.searchNearest2(hyp, r, 0)
131       *     this.searchNearest2(ref, nearestHyp, 0)
```

```
132         * (i.e. starting at 0 for every search) !
133         */
134
135       for(int ridx = 0; ridx < ref.length; ridx++)
136       {
137          int r = ref[ridx];
138          // determine boundaries of tolerance window:
139          int leftTolerance = r - this.tolerance;
140          int rightTolerance = r + this.tolerance;
141          // check, whether tolerance window boundaries are valid:
142          if(leftTolerance<0){
143             leftTolerance=0;
144          }
145          if(rightTolerance >= totalLength){
146             rightTolerance = totalLength-1;
147          }
148          int nearestHypIndex = this.searchNearest2(hyp, r, lastMatchedHyp);
149          int nearestHyp = hyp[nearestHypIndex];
150          if(nearestHyp < leftMargin || nearestHyp > rightMargin)
151          {
152             // the current hyp. boundary is either in the first half
153             // segment or in the last half, i.e. it is closer to a sequence
154             // boundary than to an actual internal reference boundary!
155             lastMatchedHyp = nearestHypIndex;
156             continue;
157          }
158          int nearestRefIndex = this.searchNearest2(ref, nearestHyp, lastMatchedRef)↩
                ;
159          int nearestRef = ref[nearestRefIndex];
160          if(nearestRef == r ) {
161             // r can be aligned with this hyp. boundary (ii):
162             res.matchedBoundaries++;
163             if(nearestHyp >= leftTolerance && nearestHyp <= rightTolerance)
164             {// this alignment can be counted as a hit (i):
165                res.truePositives++;
166                lastMatchedHyp = nearestHypIndex;
167                lastMatchedRef = nearestRefIndex;
168             }
169             // count relative hits
170             int d = Math.abs(r-nearestHyp);
171             // determine the half of the segment length in which
172             // the current hyp. boundary is located
173             double a = 0.0;
174             if(nearestHyp<r) {
175                int left = 0;
176                if(ridx>0){
177                   left = ref[ridx-1];
178                }
179                a = (double)(r - left) / 2.0;
```

```
180            } else {
181               int right = totalLength;
182               if(ridx<ref.length-1){
183                  right = ref[ridx+1];
184               }
185               a = (double)(right - r) / 2.0;
186            }
187            if(d>a) {
188               System.err.println("␣***␣WARN␣distance="+d+"␣>␣a="+a);
189            }
190            double c = (a-(double)d) / a;
191            res.relativeTP += c;
192            if(c<0) {
193               System.err.println("␣***␣WARN␣***␣ridx="+ridx
194                     +";␣ref="+r +";␣hyp="+nearestHyp
195                     +";␣a="+a +";␣d="+d + ";␣c="+c);
196            }
197            if(c>1.0) {
198               System.out.println("␣***␣WARN␣***␣ridx="+ridx
199                     +";␣ref="+r +";␣hyp="+nearestHyp
200                     +";␣a="+a +";␣d="+d + ";␣c="+c);
201            }
202         }//END:if
203      }//END:for
204   }
205
206   /**
207    * Very simple search for the boundary in seq which has the smallest distance
208    * to b. If two boundaries are equally distant, the first (left) one is chosen.
209    * @param seq
210    * @param b
211    * @param start
212    * @return the <b>index</b> of the nearest corresponding boundary of b in seq
213    */
214   private int searchNearest2(int[] seq, int b, int start) {
215      int dist = Integer.MAX_VALUE;
216      int nearestIndex = -1;
217      for(int idx = start; idx < seq.length; idx++) {
218         int h = seq[idx];
219         int d = Math.abs(h-b);
220         if(d < dist) {
221            dist = d;
222            nearestIndex = idx;
223         } else {
224            break;
225         }
226         /* no action is taken for d=dist: this means, that only the first
227          * nearest boundary is stored (there can be only two boundaries with
228          * equal distance). If d begins to increase, we have passed the
```

```
229            * closest match and can abort searching, since both lists are sorted
230            */
231        }
232        return nearestIndex;
233    }
234 }
```

Listing B.22: EvaluationResults.java

```
1  package de.uniStuttgart.danielDuran.evaluation;
2
3  /**
4   * A simple container to store the results.
5   * @author Daniel Duran
6   */
7  public class EvaluationResults {
8
9     public final String tag;
10    public final int totalFrames;
11    public final int referenceLength;
12    public final int hypothesisLength;
13    public final int tolerance;
14
15    public EvaluationResults(
16          String tag,
17          int totalFrames,
18          int refLen,
19          int hypLen,
20          int tolerance )
21    {
22       this.tag = tag;
23       this.totalFrames = totalFrames;
24       this.referenceLength  = refLen;
25       this.hypothesisLength = hypLen;
26       this.tolerance = tolerance;
27    }
28
29    /**
30     * the total number of matched boundaries, irrespective of their distance
31     */
32    public int matchedBoundaries;
33
34    /**
35     * the total number of matched and aligned boundaries (i.e. the number of
36     * matched boundaries with a distance within the defined tolerance)
37     */
38    public int truePositives;
39
40    /**
```

```
41     * the true positive count weighted by the relative distance of the
42     * hypothesized boundary to its matching reference boundary
43     */
44    public double relativeTP;
45
46    public double relativePrecision;
47    public double relativeRecall;
48    public double relativeF1;
49    public int k;
50    public double pk;
51    public double windowDiff;
52    public int falseNegatives;
53    public int falsePositives;
54    public double precision;
55    public double recall;
56    public double F1;
57    public double oversegmentation;
58    public double SERbar;
59    public double R;
60 }
```

Listing B.23: Fmeasure.java

```
1 package de.uniStuttgart.danielDuran.evaluation;
2
3 public class Fmeasure {
4
5    /**
6     * The F_{beta}-measure is defined according to
7     * "Introduction to Information Retrieval"
8     * by Christopher D. Manning, Prabhakar Raghavan and Hinrich Schuetze,
9     * 2008; Chap. 8; p. 156
10    * http://nlp.stanford.edu/IR-book/information-retrieval-book.html
11    * @param beta
12    * @return the F-measure
13    */
14    public static double getF(double beta, double precision, double recall) {
15       return ( (beta*beta + 1.0) * precision * recall)
16             / ( (beta*beta*precision) + recall  );
17    }
18
19    /**
20     * @param precision
21     * @param recall
22     * @return the F-measure with beta=1
23     * @see getF
24     */
25    public static double getF1( double precision, double recall) {
26       return getF(1.0, precision, recall);
```

```
27     }
28 }
```

B.1.2. Whole-Sequence segmentation experiments

Experiment 1: Constant-shift corpora

Listing B.24: runExperiment1.m

```
1  %%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
2  %%% file created: 2010-04-01
3  %%% run boundariness experiment 1: constant-shift artificial data
4  addpath('functions');
5  % Output base directory:
6  % (experiment1 defines a sub-directory for each run)
7  outDir = '/mount/corpora12/a2/duran_working/diss/segmentation/' ;
8  % ------------------------------------------------------------------
9  % The total length of the corpus:
10 corpusLength  = 1000003;
11 % The number of dimensions of the corpus and the probes:
12 corpusDims    = 12;
13 lambda        = 1;
14 % The range of probe lengths:
15 probeLengths  = 1:100;
16 doStoreCorpus = false;
17 doPrintGraphs = true;
18 % The number of repeated runs on a given parameter combination:
19 numRuns = 1;
20 % The location of the seed segment:
21 seedLoc = 1;
22 % ------------------------------------------------------------------
23 % The set of target lengths:
24 seedLengths = [17 31];
25 % The corresponding set of shifts:
26 seedShifts  = [97 97];
27 for rx = 1:length(seedLengths),
28     seedLength = seedLengths(rx);
29     seedShift  = seedShifts(rx);
30     outBase = [outDir, 'Exp1_', num2str(seedLength), '_', num2str(seedShift), '/'];
31     useRandomProbe = false;
32     for n = 1:numRuns,
33         experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
               seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
               lambda );
34     end
35     outBase = [outDir, 'Exp1_rnd_', num2str(seedLength), '_', num2str(seedShift), '↩
           /'];
```

```
    useRandomProbe = true;
    for n = 1:numRuns,
        experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
           lambda );
    end
end % for rx
% ------------------------------------------------------------------
% Some values according to the IMS:SWMS corpus:
% Total frames = 10267026 (MFCC)
corpusLength = 10267026;
%   phon mean-length mean-distance
%   a:      57.740     4279.790
%   d       29.265     2038.993
%   s       45.160     1683.317
%   Qu:     82.023   113432.391
seedLengths = [29];%[  58   29   45     82];
seedShifts  = [2039];%[4280 2039 1683 113432];
for rx = 1:length(seedLengths),
    seedLength = seedLengths(rx);
    seedShift  = seedShifts(rx);
    outBase = [outDir, 'Exp1_', num2str(seedLength), '_', num2str(seedShift), '/'];
    useRandomProbe = false;
    for n = 1:numRuns,
        experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
           lambda );
    end
    outBase = [outDir, 'Exp1_rnd_', num2str(seedLength), '_', num2str(seedShift), '↩
       /'];
    useRandomProbe = true;
    for n = 1:numRuns,
        experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
           lambda );
    end
end % for rx
% ------------------------------------------------------------------
% Some values according to the PL-BOSS (2006A) corpus:
% Total frames = 413987 (MFCC)
corpusLength = 413987;
% phon mean-length mean-distance
% a    29.627      649.706
% e    32.279      535.457
% r    32.110     1265.214
seedLengths = [32];%[ 30  32   32];
seedShifts  = [535];%[650 535 1265];
for rx = 1:length(seedLengths),
    seedLength = seedLengths(rx);
```

```
    seedShift  = seedShifts(rx);
    outBase = [outDir, 'Exp1_' num2str(seedLength), '_', num2str(seedShift), '/'];
    useRandomProbe = false;
    for n = 1:numRuns,
        experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
           lambda );
    end
    outBase = [outDir, 'Exp1_rnd_', num2str(seedLength), '_', num2str(seedShift), '↩
       /'];
    useRandomProbe = true;
    for n = 1:numRuns,
        experiment1( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShift, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, ↩
           lambda );
    end
end % for rx
```

Listing B.25: bndSettings.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2009-07-29

%%% Use compact format for console/log output:
format compact;
corpusScaleFactor = 100;

%%% replace 0 with NaN in boundariness sequences?
%%% ( used in bnd_getSimilarity() )
doNaN = false;

%%% Experiment 3:
% NaN frame in the memory sequence are replaced by random values with mean
% zero and the following standard deviation:
nanZeroSd = 0.001;
% This way, the original segmentation and label annotation of the corpus
% does not need to be adapted for the experiment (as the altnernative would
% be to remove NaN frames from the sequence).

%%% %%% %%% %%% %%% %%% %%% %%% %%% %%% %%% %%%
%%% %%% visualization of the results    %%% %%%

%%% which label should be assigned to the non-target parts of the corpus:
baseLabel   = 1;
targetLabel = 2;

titleFontSize = 14; % default size is 10!
xFontSize     = 16; % x label
yFontSize     = 16; % y label
```

```
labelFontSize = 30; % segment labels in the graph
tickFontSize  = 20; % tick mark labels

%%% line widths and colors:
zeroLW = 3; % zero line width
zeroLS = '--';% zero line style
zeroCol = [0.5 0.5 0.5];% zero line color
plotLW = 4.5;
plotLS = '-';
plotCol = [0.0 0.0 0.6];
labelLW = 3;
labelLS = ':';
labelCol = [0.25 0.25 0.25];
doPrintLabelText = true; %% print each segment label?
printFormat = '-depsc'; %% EPS color image

doPrintText              = false ;
doPrintCorpus            = false ;
doPrintRawSim            = true ;
doPrintRawSimDelta       = true ;
doPrintRawSimDeltaDelta  = true ;
doPrintNormSim           = true ;
doPrintNormSimDelta      = true ;
doPrintNormSimDeltaDelta = true ;
doPrintRawHits           = false ;
doPrintRawRatio          = false ;
doPrintNormHits          = false ;
doPrintNormRatio         = false ;
doPrintNormRatioDelta    = false ;
doPrintNormStd           = true ;
doPrintNormStdDelta      = false ;
doPrintRawStd            = true ;
doPrintFullNormSim       = true ;
doPrintFullRawSim        = true ;
doPrintFullNormSimDelta  = true ;
doPrintFullRawSimDelta   = true ;
```

Listing B.26: functions/experiment1.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2009-07-29
function experiment1( ouBase, corpusLength, corpusDims, ...
    seedLoc, seedLength, seedShift, probeLengths, ...
    doStoreCorpus, doPrintGraphs, useRandomProbe, lambda )
% ---------------------------------------------------------------------------
% Load global settings and initialize the simulation environment:
bndSettings;
timestamp = datestr(now,'yyyy-mm-dd+HH-MM-SS');
outDir    = [ouBase timestamp '/'];
```

```
if exist(outDir, 'dir'),
    sfx =1;
    od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    while exist(od, 'dir'),
        sfx = sfx+1;
        od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    end
    outDir = od;
end
mkdir(outDir);
% Initialize log file and start logging:
prefix        = 'Exp1_' ;
diaryFilename = [outDir, prefix, timestamp, '.diary'];
diary(diaryFilename);
fprintf('[%s] start EXPERIMENT 1...\n', datestr(now,31) );
fprintf('                    Saving diary to %s\n', diaryFilename);
fprintf('                    Output base directory created in: %s\n', outDir);
% ---------------------------------------------------------------------------
% Create corpus:
[ MS, labels, lex, seed ] = createConstantShiftMS( corpusLength, corpusDims, corpusScaleFactor, baseLabel, targetLabel, seedLoc, seedLength, seedShift, doStoreCorpus, outDir );
% ---------------------------------------------------------------------------
% Create full-length probe:
if useRandomProbe,
    probe = randn(probeLengths(end),corpusDims)  .* corpusScaleFactor ;
    disp(['[', datestr(now,31), '] using random probe', ]);
else
    probe = createProbe(seed, probeLengths(end), corpusDims);
end
if doStoreCorpus,
    outFileName = [outDir,prefix, 'probe.txt'];
    save(outFileName,'MS', '-ASCII');
    disp(['[', datestr(now,31), '] saved probe to file ', outFileName ]);
end
%%% run comparison
fprintf('[%s] Computing similarity...\n', datestr(now,31));
[totalSimilarity, totalSimilarityN, rawHits, normHits, rawStd, normStd, fullRawSim, fullNormSim] = ...
    getSimilarity(MS, probeLengths, probe, lambda);
fprintf('[%s] ...done\n', datestr(now,31));
% Print some stats about the results:
fullRawSimDelta  = diff(fullRawSim);
fullNormSimDelta = diff(fullNormSim);
fprintf('full raw similarity:   min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(fullRawSim), max(fullRawSim), mean(fullRawSim),  std(fullRawSim) );
fprintf('full raw sim. delta:   min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(fullRawSimDelta), max(fullRawSimDelta), mean(fullRawSimDelta),  std(fullRawSimDelta) );
```

```
fprintf('full norm. similarity: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(fullNormSim), max(fullNormSim), mean(fullNormSim),  std(fullNormSim) );
fprintf('full norm. sim. delta: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(fullNormSimDelta), max(fullNormSimDelta), mean(fullNormSimDelta),  std(fullNormSimDelta) );
totalSimilarityDelta = diff(totalSimilarity);
totalSimilarityNDelta = diff(totalSimilarityN);
fprintf('raw similarity:   min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarity), max(totalSimilarity), mean(totalSimilarity),  std(totalSimilarity) );
fprintf('raw sim. delta:   min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarityDelta), max(totalSimilarityDelta), mean(totalSimilarityDelta),  std(totalSimilarityDelta) );
fprintf('norm. similarity: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarityN), max(totalSimilarityN), mean(totalSimilarityN),  std(totalSimilarityN) );
fprintf('norm. sim. delta: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarityNDelta), max(totalSimilarityNDelta), mean(totalSimilarityNDelta), std(totalSimilarityNDelta) );
%%% store Results
outFileName = [outDir,prefix, 'rawSim.txt'];
save(outFileName,'totalSimilarity', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir,prefix,  'normSim.txt'];
save(outFileName,'totalSimilarityN', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir,prefix,  'rawStd.txt'];
save(outFileName,'rawStd', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir,prefix,  'normStd.txt'];
save(outFileName,'normStd', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir,prefix,  'fullRawSim.txt'];
save(outFileName,'fullRawSim', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir,prefix,  'fullNormSim.txt'];
save(outFileName,'fullNormSim', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
%%% evaluate : print graphs
if doPrintGraphs,
    printAllGraphs( outDir, MS(seedLoc:probeLengths(end)), totalSimilarity, totalSimilarityN, ...
        rawHits, rawStd, normHits, normStd, fullNormSim, fullRawSim, ...
        probeLengths, timestamp, labels, lex, seedLoc, prefix );
else
    disp(['[', datestr(now,31), '] (skipped printing graphics)']);
end
%%% END
```

```
fprintf('[%s] %s: All done.\nStart time was: %s\nBye!\n',  datestr(now,31), mfilename, timestamp );
diary off;
end
```

Listing B.27: functions/createConstantShiftMS.m

```
function [ MS, labels, lex, target ] = createConstantShiftMS( corpusLength, corpusDims, corpusScaleFactor, baseLabel, targetLabel, seedLoc, seedLength, seedShift, doStoreCorpus, outDir )
% Create a "constant-shift corpus" of random vectors and a repeated target
% segment at constant intervals.
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
% ---------------------------------------------------------------------------
% Create corpus:
% here, a random "signal" based on the standard normal distribution
MS = randn(corpusLength,corpusDims) * corpusScaleFactor ;
% normalize corpus if corpusDims > 1
if corpusDims > 1,
    for idx = 1:length(MS) ,
        n = norm(MS(idx,:));
        MS(idx,:) = MS(idx,:) ./ n ;
    end
end
% Define and initialize segment labels:
lex    = [{'...'}; {'X'}]; %%% == segName
labels = repmat(baseLabel, corpusLength,1);%% == segmentLabels
target     = MS(seedLoc:(seedLoc+seedLength-1),:);
targetLocs = []; %%% == segLocs
% put targets into corpus and create the correpsonding labels:
for idx = seedLoc:seedShift:corpusLength,
    if idx <= corpusLength-(seedLength-1),
        MS(idx:idx+seedLength-1,:) = target;
        labels(idx:idx+seedLength-1) = targetLabel;
        targetLocs = [targetLocs; idx];
    end
end
% ---------------------------------------------------------------------------
% Print some information about the memory sequence corpus:
disp(['[', datestr(now,31), '] test corpus initialized: ']);
disp(['   size=', num2str(size(MS,1)), 'x', num2str(size(MS,2))]);
if ( size(targetLocs,1)<10 ),
    maxTargetLocs = size(targetLocs,1);
else
    maxTargetLocs = 10;
end
disp([' target size =', num2str(size(target,1)), 'x', num2str(size(target,2))]);
disp([' target shift=', num2str(seedShift) ]);
```

```
disp(['␣targets:␣', num2str(size(targetLocs,1)), ';␣locations␣1-10:␣[', num2str(↩
    targetLocs(1:maxTargetLocs)'), ']' ]); %'
% ---------------------------------------------------------------------------
% Store corpus to file
if doStoreCorpus,
    outFileName = [outDir,prefix, 'corpus.txt'];
    save(outFileName,'MS', '-ASCII');
    disp(['[', datestr(now,31), ']␣saved␣corpus␣to␣file␣', outFileName ]);
    outFileName = [outDir,prefix, 'targetLocs.txt'];
    save(outFileName,'targetLocs', '-ASCII');
    disp(['[', datestr(now,31), ']␣saved␣corpus␣to␣file␣', outFileName ]);
    outFileName = [outDir,prefix, 'labels.txt'];
    save(outFileName,'labels', '-ASCII');
    disp(['[', datestr(now,31), ']␣saved␣corpus␣to␣file␣', outFileName ]);
end
end
```

Listing B.28: functions/createProbe.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2010-08-25 (taken from bnd_getSimilarity.m)
function probe = createProbe( target, maxLen, dims )
%CREATEPROBE creates a full-length probe
bndSettings;

%%% create the probe such that for probe lengths > target length
%%% a random value is appended to the current probe
%%% --> this guarantees that there is, very likely, no corresponding
%%% sequence in the corpus
if maxLen > size(target,1),
    appendix = randn(maxLen-size(target,1),dims) .* corpusScaleFactor;
    if dims > 1,
        for idx = 1:length(appendix) ,
            n = norm(appendix(idx,:));
            appendix(idx,:) = appendix(idx,:) ./ n ;
        end
    end
    probe = [target; appendix];
else
    probe = target(1:maxLen,:) ;
    warning('The␣probe␣will␣be␣shorter␣than␣the␣target␣segment!');
end
end
```

Listing B.29: getSimilarity.c

```
/* Daniel Duran, SFB 732/A2, IMS, Uni Stuttgart
 * "boundariness"
 *
```

```
4  * This is a MEX-file for MATLAB.
5  * File created: 2010-08-16
6  * compile:
7  * mex getSimilarity.c CFLAGS="\$CFLAGS -fopenmp" LDFLAGS="\$LDFLAGS -fopenmp"
8  */
9 #include "mex.h"
10 #include <math.h>
11 #include <stdlib.h>
12 #include <time.h>
13 #include <omp.h>
14
15 void getSimilarity(double *corpus, int corpusSize1,
16                    double *probeLengths, int probeLengthsSize2,
17                    double *probe, int probeSize1,
18                    int featureDims, int lambda,
19                    double *totalSimilarity, double *totalSimilarityN,
20                    double *rawHits, double *normHits, double *rawStd, double *↩
                         normStd,
21                    double *fullRawSim, double *fullNormSim );
22
23 /* ********************************************************************** *
24  *                            cross correlation                           *
25  * ********************************************************************** *
26    input: corpus    == Array (corpusSize1 x cols)
27           thisProbe == Array (probeSize1 x cols)
28  */
29 int myxcorr(double * corpus, int corpusSize1, double * thisProbe, int probeSize1,
30             int pLen, int start, int cols,
31             double * out, double * outN)
32 {
33   int j, dim;
34   double sum, x2, y2;
35
36   double c_n_m;
37   double p_n_m;
38
39   out[start]  = 0.0;
40   outN[start] = 0.0;
41
42   for(dim = 0; dim < cols; dim++)/* PAPER: l = 1:12 */
43   {
44     sum = 0.0;
45     x2 = 0.0;
46     y2 = 0.0;
47
48     for(j = 0; j < pLen; j++) /* PAPER: i=m:n */
49     {
50       /*>>> corpus[n][m] = corpus[(n-1) + ((m-1)*N) ] */
51       c_n_m = corpus[start+j + (dim*corpusSize1)];
```

```
52        p_n_m = thisProbe[j + (dim*probeSize1)];
53
54        sum = sum + (c_n_m * p_n_m);
55
56        y2  = y2 + (c_n_m * c_n_m);
57        x2  = x2 + (p_n_m * p_n_m);
58      }
59      out[start]  = out[start] + sum;
60      outN[start] = outN[start] + (sum / sqrt(x2 * y2));/* <- <math.h> */
61    }
62
63    return 0;
64  }/* END myxcorr */
65
66  /* ********************************************************************** *
67   *                            standard deviation                          *
68   * ********************************************************************** */
69  double myStd(double * matrix, double  mean, int rows)
70  {  /* ----------------------------------------------------------------
71         std = sqrt( SUM( (X - M)^2 )  / N - 1 ) -- or: just ".../ N" ?
72
73         with s^2 = variance
74              M   = sample mean
75              N   = size of sample
76        ---------------------------------------------------------------- */
77     int i;
78     double sum=0.0;
79     double out;
80     for(i=0; i < rows; i++)
81     {
82        sum = sum + ( (matrix[i] - mean) * (matrix[i] - mean) );
83     }
84     out = sqrt( sum / (rows - 1) );
85     return out;
86  }/* END myColumnStd */
87
88  /* ********************************************************************** *
89   *                              get hits                                  *
90   *                                                                        *
91   *   compute A - the "above threshold values" and their standard deviation  *
92   * ********************************************************************** */
93  int getHits(double * matrix, int rows, double th,
94              int idx,
95              double * out, double * outRows, double * outStd )
96  {
97     int i;
98     double sum=0.0;
99     double mean = 0.0;
100    double sumx2=0.0;
```

```
101    double * x2;
102    int * check;
103    x2 = (double *) mxMalloc(rows * sizeof(double));
104    check = (int *) mxMalloc(rows * sizeof(int));
105
106    int hits=0;
107
108    for(i=0; i < rows; i++){
109       if(matrix[i] > th)
110       {
111          hits++;
112          sum = sum + matrix[i];
113          check[i] = 1;
114       } else {
115          check[i] = 0;
116       }
117    }
118
119    mean = sum / hits;
120
121    #pragma omp parallel for shared(rows,matrix,mean,check,x2) private(i) schedule(↩
          dynamic,6)
122    for(i=0; i<rows; i++){
123       if(check[i]==1){
124          x2[i] = (matrix[i]-mean) * (matrix[i]-mean) ;
125       } else {
126          x2[i] = 0;
127       }
128    }
129
130    for(i=0; i<rows; i++){
131       if(check[i]==1){
132          sumx2 = sumx2 + x2[i];
133       }
134    }
135
136    out[idx] = (sum / hits); /* total similarity */
137    outStd[idx] = sqrt( sumx2 / (hits - 1) );
138    outRows[idx] = hits;
139
140    return 0;
141 }
142 /* ==================================================================== */
143
144 /* ==================================================================== *
145  * "The gateway function"                                               *
146  *                                                                      *
147  * MATLAB call:                                                         *
148  * [totalSimilarity, totalSimilarityN, rawHits, normHits, ...           *
```

```
149  *  rawStd, normStd, fullRawSim, fullNormSim] = ...                          *
150  *                       bnd_getSimilarity(MS, probeLengths, probe, lambda); *
151  * ========================================================================== */
152 void mexFunction( int nlhs, mxArray *plhs[],
153                   int nrhs, const mxArray *prhs[])
154 {
155    /* IN */
156    double    *corpus;
157    int        corpusDims;
158    const int *corpusSize;
159    int        corpusCnt;
160    double    *probeLengths;
161    int        probeLengthsDims;
162    const int *probeLengthsSize;
163    int        probeLengthsCnt;
164    double    *probe;
165    int        probeDims;
166    const int *probeSize;
167    int lambda;
168    int featureDims; /* the number of feature dimensions of the input data, e.g. 13 ↩
          for MFCC */
169
170    /* OUT */
171    int *simSize;
172    double *totalSimilarity;
173    double *totalSimilarityN;
174    double *rawHits;
175    double *normHits;
176    double *rawStd;
177    double *normStd;
178    double *fullRawSim;
179    double *fullNormSim;
180
181    /* === check input parameters === */
182    if(nrhs != 4){
183       mexErrMsgTxt("Four␣input␣parameters␣required!\n");
184    }
185
186    /* corpus = MS */
187    corpus     = mxGetPr(prhs[0]);
188    corpusDims = mxGetNumberOfDimensions(prhs[0]);
189    corpusSize = mxGetDimensions(prhs[0]);
190
191    corpusCnt  = mxGetNumberOfElements(prhs[0]);
192
193    /* probeLengths */
194    probeLengths     = mxGetPr(prhs[1]);
195    probeLengthsDims = mxGetNumberOfDimensions(prhs[1]);
196    probeLengthsSize = mxGetDimensions(prhs[1]);
```

```
197
198   probeLengthsCnt  = mxGetNumberOfElements(prhs[1]);
199
200   /* probe */
201   probe     = mxGetPr(prhs[2]);
202   probeDims = mxGetNumberOfDimensions(prhs[2]);
203   probeSize = mxGetDimensions(prhs[2]);
204
205   /* lambda */
206   lambda = (int)*mxGetPr(prhs[3]);
207
208   if(corpusDims != 2 || probeDims != 2){
209      mexErrMsgTxt("corpus␣and␣probe␣must␣be␣2-dimensional␣matrices!\n");
210   }
211
212   if(corpusSize[1] != probeSize[1]){
213      mexErrMsgTxt("corpus␣and␣probe␣must␣have␣an␣equal␣number␣of␣feature␣↩
         dimenstions!\n(i.e.␣corpus(N␣x␣D)␣and␣probe(K␣x␣D))\n");
214   }
215
216   featureDims = corpusSize[1];
217
218   /* === initialize output parameters === */
219   simSize    = (int *)mxMalloc(2*sizeof(int));
220   simSize[0] = probeLengths[ probeLengthsSize[1]-1 ]; /* max length */
221   simSize[1] = 1;
222
223   plhs[0]          = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
224   totalSimilarity  = mxGetPr(plhs[0]);
225
226   plhs[1]          = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
227   totalSimilarityN = mxGetPr(plhs[1]);
228
229   plhs[2]  = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
230   rawHits  = mxGetPr(plhs[2]);
231
232   plhs[3]  = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
233   normHits = mxGetPr(plhs[3]);
234
235   plhs[4] = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
236   rawStd  = mxGetPr(plhs[4]);
237
238   plhs[5] = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
239   normStd = mxGetPr(plhs[5]);
240
241   plhs[6]     = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
242   fullRawSim  = mxGetPr(plhs[6]);
243
244   plhs[7]     = mxCreateNumericArray(2, simSize, mxDOUBLE_CLASS, mxREAL);
```

```
245    fullNormSim = mxGetPr(plhs[7]);
246
247    getSimilarity(corpus, corpusSize[0], probeLengths, probeLengthsSize[1],
248        probe, probeSize[0], featureDims, lambda,
249        totalSimilarity, totalSimilarityN, rawHits, normHits, rawStd, normStd,
250        fullRawSim, fullNormSim);
251 }
252 /* ========================================================================== */
253
254
255 /* ========================================================================== */
256 void getSimilarity(double *corpus, int corpusSize1,
257                    double *probeLengths, int probeLengthsSize2,
258                    double *probe, int probeSize1,
259                    int featureDims, int lambda,
260                    double *totalSimilarity, double *totalSimilarityN,
261                    double *rawHits, double *normHits, double *rawStd, double *↩
                          normStd,
262                    double *fullRawSim, double *fullNormSim  )
263 {
264    double * xcorrSeries;
265    /* the "raw correlation score" series
266          -> the local similarity "r" at each index in the corpus */
267
268    double * xcorrNSeries;
269
270    double rawSum;
271    double normSum;
272    double rawMean;
273    double normMean;
274    double rawLenStd;
275    double normLenStd;
276    double rawTh;
277    double normTh;
278
279    int actual_rows;
280    int minLength;   /* the minimum probe length */
281    int maxLength;   /* the maximum probe length */
282    int lastIndex;
283
284    int pLen, i, j; /* loop vars */
285
286    actual_rows = corpusSize1;
287    minLength = probeLengths[ 0 ];
288    maxLength = probeLengths[ probeLengthsSize2-1 ];
289
290    /* == allocate memory */
291    xcorrSeries  = (double*) mxMalloc(actual_rows * sizeof(double));
292    xcorrNSeries = (double*) mxMalloc(actual_rows * sizeof(double));
```

```
293
294    for(pLen = minLength; pLen <= maxLength; pLen++)
295    {
296       lastIndex = actual_rows - pLen + 1;
297
298       #pragma omp parallel for shared(corpus, corpusSize1, probe, probeSize1, pLen,↩
              featureDims, xcorrSeries, xcorrNSeries) private(i) schedule(dynamic,6)
299       for(i = 0; i < lastIndex; i++){
300          myxcorr(corpus, corpusSize1, probe, probeSize1, pLen, i, featureDims, ↩
                xcorrSeries, xcorrNSeries);
301       }
302
303       rawSum = 0.0;
304       normSum = 0.0;
305       for(i=0; i < lastIndex; i++){
306          rawSum = rawSum + xcorrSeries[i];
307          normSum = normSum + xcorrNSeries[i];
308       }
309
310       rawMean  = rawSum / lastIndex;
311       normMean = normSum / lastIndex;
312
313       rawLenStd  = myStd(xcorrSeries, rawMean, lastIndex);
314       normLenStd = myStd(xcorrNSeries, normMean, lastIndex);
315
316       rawTh  = rawMean + (lambda * rawLenStd);/* raw threshold */
317       normTh = normMean + (lambda * normLenStd);
318
319       fullRawSim[pLen-1] = rawSum;
320       fullNormSim[pLen-1]= normSum;
321
322       getHits(xcorrSeries, lastIndex, rawTh, pLen-1, totalSimilarity, rawHits, ↩
             rawStd);
323       getHits(xcorrNSeries, lastIndex, normTh, pLen-1, totalSimilarityN, normHits, ↩
             normStd);
324
325    }/* END for pLen */
326 }
```

Listing B.30: functions/printAllGraphs.m

```
1 %%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
2 %%% file created: ?
3 %%% edited:       2010-08-13 ...
4 function printAllGraphs( outDir, MSsegment, totalSimilarity, totalSimilarityN, ...
5     rawHits, rawStd, normHits, normStd, fullNormSim, fullRawSim, ...
6     probeLengths, timestamp, labels, lex, probeLoc, prefix )
7 %PRINTALLGRAPHS wrapper function to call printCurves
8
```

```
%%% load global settings
bndSettings;

%%% text:
title = '';
xText = '';
yText = '';
startLen = probeLengths(1);
endLen   = probeLengths(end);

if doPrintCorpus,
    if doPrintText ,
        title = 'Corpus segment';
        xText = '';
        yText = '';
    end
    fileName = [prefix, timestamp, '_corpus'];
    printCurves(outDir, MSsegment, startLen, endLen, fileName, title, labels, lex, ↩
       probeLoc, xText, yText);
end
if doPrintRawSim,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'raw cumul. similarity';
    end
    fileName = [prefix, timestamp, '_rawSim'];
    printCurves(outDir, totalSimilarity, startLen, endLen, fileName, title, labels,↩
        lex, probeLoc, xText, yText);
end
if doPrintRawSimDelta,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'first deriv. of raw cumul. similarity';
    end
    fileName = [prefix, timestamp, '_rawSimDelta'];
    printCurves(outDir, diff(totalSimilarity), startLen, endLen, fileName, title, ↩
       labels, lex, probeLoc, xText, yText);
end
if doPrintRawSimDeltaDelta,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'second deriv. of raw cumul. similarity';
    end
    fileName = [prefix, timestamp, '_rawSimDeltaDelta'];
    printCurves(outDir, diff(diff(totalSimilarity)), startLen, endLen, fileName, ↩
       title, labels, lex, probeLoc, xText, yText);
```

```
end
if doPrintNormSim,
    if doPrintText ,
        title = '';
        xText = 'probe␣length';
        yText = 'norm.␣cumul.␣similarity';
    end
    fileName = [prefix, timestamp, '_normSim'];
    printCurves(outDir, totalSimilarityN, startLen, endLen, fileName, title, labels↩
       , lex, probeLoc, xText, yText);
end
if doPrintNormSimDelta,
    if doPrintText ,
        title = '';
        xText = 'probe␣length';
        yText = 'first␣deriv.␣of␣norm.␣cumul.␣similarity';
    end
    fileName = [prefix, timestamp, '_normSimDelta'];
    printCurves(outDir, diff(totalSimilarityN), startLen, endLen, fileName, title, ↩
       labels, lex, probeLoc, xText, yText);
end
if doPrintNormSimDeltaDelta,
    if doPrintText ,
        title = '';
        xText = 'probe␣length';
        yText = 'second␣deriv.␣of␣norm.␣cumul.␣similarity';
    end
    fileName = [prefix, timestamp, '_normSimDeltaDelta'];
    printCurves(outDir, diff(diff(totalSimilarityN)), startLen, endLen, fileName, ↩
       title, labels, lex, probeLoc, xText, yText);
end
if doPrintRawHits,
    if doPrintText ,
        title = '';
        xText = 'probe␣length';
        yText = 'hits';
    end
    fileName = [prefix, timestamp, '_rawHits'];
    printCurves(outDir, rawHits, startLen, endLen, fileName, title, labels, lex, ↩
       probeLoc, xText, yText);
end
if doPrintRawRatio,
    if doPrintText ,
        title = '';
        xText = 'probe␣length';
        yText = 'std␣/␣sim.';
    end
    fileName = [prefix, timestamp, '_rawRatio'];
```

```
    printCurves(outDir, (rawStd ./ totalSimilarity), startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintNormHits,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'norm. hits';
    end
    fileName = [prefix, timestamp, '_normHits'];
    printCurves(outDir, normHits, startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintNormRatio,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'norm. std / norm. sim.';
    end
    fileName = [prefix, timestamp, '_normRatio'];
    printCurves(outDir, (normStd ./ totalSimilarityN), startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintNormRatioDelta,
    if doPrintText,
        title = '';
        xText = '';
        yText = '';
    end
    fileName = [prefix, timestamp, '_normRatioDelta'];
    printCurves(outDir, diff((normStd ./ totalSimilarityN)), startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintNormStd,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'norm. sim. std.';
    end
    fileName = [prefix, timestamp, '_normStd'];
    printCurves(outDir, normStd, startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintNormStdDelta,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'first deriv. of norm. sim. std.';
    end
```

```
    fileName = [prefix, timestamp, '_normStdDelta'];
    printCurves(outDir, diff(normStd), startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintRawStd,
    if doPrintText ,
        title = '';
        xText = 'probe length';
        yText = 'raw sim. std.';
    end
    fileName = [prefix, timestamp, '_rawStd'];
    printCurves(outDir, rawStd, startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintFullNormSim,
    if doPrintText,
        title = '';
        xText = '';
        yText = 'full norm. sim.';
    end
    fileName = [prefix, timestamp, '_fullNormSim'];
    printCurves(outDir, fullNormSim, startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintFullRawSim,
    if doPrintText,
        title = '';
        xText = '';
        yText = 'full raw sim.';
    end
    fileName = [prefix, timestamp, '_fullRawSim'];
    printCurves(outDir, fullRawSim, startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintFullNormSimDelta,
    if doPrintText,
        title = '';
        xText = '';
        yText = 'full norm. sim. delta';
    end
    fileName = [prefix, timestamp, '_fullNormSimDelta'];
    printCurves(outDir, diff(fullNormSim), startLen, endLen, fileName, title, labels, lex, probeLoc, xText, yText);
end
if doPrintFullRawSimDelta,
    if doPrintText,
        title = '';
        xText = '';
        yText = 'full raw sim. delta';
```

```
    end
    fileName = [prefix, timestamp, '_fullRawSimDelta'];
    printCurves(outDir, diff(fullRawSim), startLen, endLen, fileName, title, labels↩
        , lex, probeLoc, xText, yText);
end
end
```

Listing B.31: functions/printCurves.m

```
function printCurves(outDir, data, startLen, endLen, dateinameSuffix, titleText, ↩
   labels, lex, probeLoc, xText, yText)
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2009-07-29

%%% load global settings
bndSettings;

skip = false;
if sum(isnan(data)) == size(data,1) ,
    skip = true;
    disp(['[', datestr(now,31), '] <', mfilename, '> ! skipped printing empty (NaN)↩
        data for "', dateinameSuffix, '"']);
end
if size(data,2) > 1,
    skip = true;
    disp(['[', datestr(now,31), '] <', mfilename, '> ! skipped printing matrix with↩
        ', num2str(size(data,2)), ' columns "', dateinameSuffix, '"']);
end
if min(data) == max(data),
    skip = true;
    disp(['[', datestr(now,31), '] <', mfilename, '> ! skipped printing matrix with↩
        min (', num2str(min(data)), ) == max (', num2str(max(data)), )']);
end
if ~skip,
    %%% DEBUG / INFO
    disp(['[', datestr(now,31), '] <', mfilename, '> printing results for probe ↩
        lengths ', num2str(startLen), '-', num2str(endLen), ' of probe index ', ↩
        num2str(probeLoc) ]);
    %%% plot graph
    figure;
    hold on;
    % adjust x-axis:
    xlim([0, endLen+1]);
    xl = xlim;
    % adjust y-axis:
    r = max(data) - min(data);
    ylim([min(data) - r*0.1, max(data) + r*0.1]);
    yl = ylim;
    % zero line:
```

```
    line(xlim, [0; 0], 'LineWidth',zeroLW, 'Color',zeroCol, 'LineStyle',zeroLS);
    box on;
    plot(data, 'Color',plotCol, 'LineStyle',plotLS, 'LineWidth',plotLW);
    % graph title
    if ~strcmp(titleText, ''),
        title(titleText, 'FontSize', titleFontSize);
    end
    % axes labels
    if(~strcmp(xText,'')),
        xlabel(xText, 'FontSize', xFontSize);
    end
    if(~strcmp(yText,'')),
        ylabel(yText, 'FontSize', yFontSize);
    end
    % get handle to current axes
    h = gca;
    set(h, 'FontSize', tickFontSize);
    yLabelPos = yl(2) - ((yl(2) - yl(1)) * 0.05);
    % overlay the labels:
    tmpSegLab1 = 0;
    for idx = xl(1):xl(2),
        if idx > 0,
            tmpSegLab2 = labels(idx+probeLoc);
            if(tmpSegLab2 ~= tmpSegLab1),
                line([idx; idx], ylim', 'LineWidth',labelLW, 'Color',labelCol, '↩
                   LineStyle',labelLS); %'
                tmpSegLab1 = tmpSegLab2;

                if doPrintLabelText && idx < (xl(2)-5) && labels(idx+probeLoc) > 0,
                    text( idx + 1, yLabelPos, char(lex(labels(idx+probeLoc))), '↩
                       Color',labelCol, 'FontName','FixedWidth', 'FontSize', ↩
                       labelFontSize);
                end
            end %%% if
        end
    end %%% idx
    %%% save to file:
    dateiname = [outDir,  dateinameSuffix ];
    disp(['[', datestr(now,31), '] <', mfilename, '> saving ', dateiname, '.eps']);↩
       % INFO
    print(printFormat, dateiname);% Grafik in Datei schreiben
    hold off;
    close(gcf);
end
```

Experiment 2: Variable-shift corpora

This sections lists the source code of the experiment described in section 4.4.2, starting on page 143.

Listing B.32: runExperiment2.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2010-02-26
%%% run boundariness experiment 2: variable-shift artificial data
addpath('functions');
outDir = '/mount/corpora12/a2/duran_working/diss/segmentation/' ;
corpusLength  = 1000003;
corpusDims    = 12;
lambda        = 1;
probeLengths  = 1:100;
doStoreCorpus = false;
doPrintGraphs = true;
numRuns       = 1;
seedLoc       = 1;
% The set of target lengths:
seedLengths       = [11 17 31];
% The corresponding set of shifts:
seedShiftFactors  = [29 97 97];
for rx = 1:length(seedLengths),
    seedLength         = seedLengths(rx);
    seedShiftMaxFactor = seedShiftFactors(rx);
    outBase = [outDir, 'Exp2_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = false;
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, lambda );
    end
    outBase = [outDir, 'Exp2_rnd_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = true;
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, lambda );
    end
end % for rx
% ------------------------------------------------------------------
% Some values according to the IMS:SWMS corpus:
% Total frames = 10267026 (re-sampled)
corpusLength = 10267026;
% phon mean-length mean-distance  dist/len
% a:      57.740   4279.790       74.122
```

```
% d       29.265   2038.993       69.673
% s       45.160   1683.317       37.275
% Qu:     82.023 113432.391     1382.934
seedLengths      = [29]; %[58 29 45   82];
seedShiftFactors = [138]; %[74 69 37 1383];
for rx = 1:length(seedLengths),
    seedLength         = seedLengths(rx);
    seedShiftMaxFactor = seedShiftFactors(rx);
    outBase = [outDir, 'Exp2_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = false;
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, lambda );
    end
    outBase = [outDir, 'Exp2_rnd_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = true;
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, lambda );
    end
end % for rx
% ------------------------------------------------------------------
% Some values according to the PL-BOSS (2006A) corpus:
% Total frames = 413987 (re-sampled)
corpusLength = 413987;
% phon mean-length mean-distance  dist/len
% a    29.627      649.706        21.930
% e    32.279      535.457        16.588
% r    32.110     1265.214        39.402
seedLengths       = [32]; %[30 32 32];
seedShiftFactors  = [34]; %[22 17 39];
for rx = 1:length(seedLengths),
    seedLength         = seedLengths(rx);
    seedShiftMaxFactor = seedShiftFactors(rx);
    outBase = [outDir, 'Exp2_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = false;
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, useRandomProbe, lambda );
    end
    outBase = [outDir, 'Exp2_rnd_', num2str(seedLength), '_x', num2str(seedShiftMaxFactor), '/'];
    useRandomProbe = true;
```

```
    for n = 1:numRuns,
        experiment2( outBase, corpusLength, corpusDims, seedLoc, seedLength, ↩
           seedShiftMaxFactor, probeLengths, doStoreCorpus, doPrintGraphs, ↩
           useRandomProbe, lambda );
    end
end % for rx
```

Listing B.33: functions/experiment2.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2009-07-30 -- copy from bnd_experiment1.m
function experiment2( ouBase, corpusLength, corpusDims, ...
    seedLoc, seedLength, seedShiftMaxFactor, probeLengths, ...
    doStoreCorpus, doPrintGraphs, useRandomProbe, lambda )

%%% load global settings
bndSettings;
%%% logging
timestamp = datestr(now,'yyyy-mm-dd+HH-MM-SS');
outDir    = [ouBase timestamp '/'];
if exist(outDir, 'dir'),
    sfx =1;
    od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    while exist(od, 'dir'),
        sfx = sfx+1;
        od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    end
    outDir = od;
end
mkdir(outDir);
prefix = 'Exp2_' ;
diaryFilename = [outDir, prefix, timestamp, '.diary'];
diary(diaryFilename);
fprintf('[%s] start EXPERIMENT 2...\n', datestr(now,31) );
fprintf('                    Saving diary to %s\n', diaryFilename);
fprintf('                    Output base directory created in: %s\n', outDir);
fprintf('                    OMP_NUM_THREADS = %d\n', getenv('OMP_NUM_THREADS'));
%%% create corpus
[ MS, labels, lex, seed  ] = createVariableShiftMS( corpusLength, corpusDims, ↩
   corpusScaleFactor, ...
    baseLabel, targetLabel, seedLoc, seedLength, seedShiftMaxFactor, doStoreCorpus,↩
        outDir ) ;
%%% %%% %%%
if useRandomProbe,
    probe = randn(probeLengths(end),corpusDims)  .* corpusScaleFactor ;
    disp(['[', datestr(now,31), '] using random probe', ]);
else
    probe = createProbe(seed, probeLengths(end), corpusDims);
end
```

```
% MEX >>>
[totalSimilarity, totalSimilarityN, rawHits, normHits, rawStd, normStd, fullRawSim,↩
    fullNormSim] = ...
    getSimilarity(MS, probeLengths, probe, lambda);
%%% store Results
outFileName = [outDir,prefix, timestamp, '_rawSim.txt'];
save(outFileName,'totalSimilarity', '-ASCII');
disp(['[', datestr(now,31), ']␣saved␣result␣to␣file␣', outFileName ]);
outFileName = [outDir,prefix, timestamp, '_normSim.txt'];
save(outFileName,'totalSimilarityN', '-ASCII');
disp(['[', datestr(now,31), ']␣saved␣result␣to␣file␣', outFileName ]);
outFileName = [outDir,prefix, timestamp, '_rawStd.txt'];
save(outFileName,'rawStd', '-ASCII');
disp(['[', datestr(now,31), ']␣saved␣result␣to␣file␣', outFileName ]);
outFileName = [outDir,prefix, timestamp, '_normStd.txt'];
save(outFileName,'normStd', '-ASCII');
disp(['[', datestr(now,31), ']␣saved␣result␣to␣file␣', outFileName ]);
%%% evaluate : print graphs
if doPrintGraphs,
    printAllGraphs( outDir, MS(seedLoc:probeLengths(end)), totalSimilarity, ↩
       totalSimilarityN, ...
        rawHits, rawStd, normHits, normStd, fullNormSim, fullRawSim, ...
        probeLengths, timestamp, labels, lex, seedLoc, prefix );
else
    disp(['[', datestr(now,31), ']␣(skipped␣printing␣graphics)']);
end
%%% END
disp(['[', datestr(now,31), ']␣<', mfilename, '>␣all␣done.␣bye!']);
diary off;
end
```

Listing B.34: functions/createVariableShiftMS.m

```
function [ MS, labels, lex, target  ] = createVariableShiftMS( corpusLength, ↩
   corpusDims, corpusScaleFactor, baseLabel, targetLabel, seedLoc, seedLength, ↩
   seedShiftMaxFactor, doStoreCorpus, outDir )
MS = randn(corpusLength,corpusDims) * corpusScaleFactor;%% normal distribution
%%% normalize corpus if corpusDims > 1
if corpusDims > 1,
    for idx = 1:length(MS) ,
        n = norm(MS(idx,:));
        MS(idx,:) = MS(idx,:) ./ n ;
    end
end
lex      = [{'...'}; {'X'}]; %%% == segName
labels   = repmat(baseLabel, corpusLength, 1);%% == segmentLabels
target   = MS(seedLoc:(seedLoc+seedLength-1),:);
shiftArray = randi([seedLength+1, seedLength*seedShiftMaxFactor], floor(↩
   corpusLength / seedLength),1);
```

```
14 targetLocs = [];
15 %%% put targets into corpus and create the correpsonding labels:
16 labels(seedLoc:seedLoc+seedLength-1) = targetLabel;
17 targetLocs = [targetLocs; seedLoc];
18 idx = seedLoc;
19 for c = 1:length(shiftArray),
20     idx = idx  + shiftArray(c);
21     if idx <= corpusLength-(seedLength-1),
22         MS(idx:idx+seedLength-1,:) = target;
23         labels(idx:idx+seedLength-1) = targetLabel;
24         targetLocs = [targetLocs; idx];
25     end
26 end
27 % ---------------------------------------------------------------------------
28 % Print some information about the memory sequence corpus:
29 %
30 disp(['[', datestr(now,31), '] test corpus initialized: ']);
31 disp(['   size=', num2str(size(MS,1)), 'x', num2str(size(MS,2))]);
32 if ( size(targetLocs,1)<10 ),% (relevant for the logger output only)
33     maxTargetLocs = size(targetLocs,1);
34 else
35     maxTargetLocs = 10;
36 end
37 disp([' target size= ', num2str(size(target,1)), 'x', num2str(size(target,2))]);
38 disp([' target shift max. factor=', num2str(seedShiftMaxFactor) ]);
39 disp([' target shift min       =', num2str(min(shiftArray)) ]);
40 disp([' target shift max       =', num2str(max(shiftArray)) ]);
41 disp([' target shift mean      =', num2str(mean(shiftArray)) ]);
42 disp([' targets: ', num2str(size(targetLocs,1)), '; locations 1-10: [', num2str(↩
     targetLocs(1:maxTargetLocs)'), ']' ]); %'
43 %%% store corpus to file
44 if doStoreCorpus,
45     outFileName = [outDir, 'vsCorpus.txt'];
46     save(outFileName,'MS', '-ASCII');
47     disp(['[', datestr(now,31), '] saved corpus to file ', outFileName ]);
48     outFileName = [outDir, 'vsTargetLocs.txt'];
49     save(outFileName,'targetLocs', '-ASCII');
50     disp(['[', datestr(now,31), '] saved corpus to file ', outFileName ]);
51     outFileName = [outDir, 'vsLabels.txt'];
52     save(outFileName,'labels', '-ASCII');
53     disp(['[', datestr(now,31), '] saved corpus to file ', outFileName ]);
54 end
55 end
```

Experiment 3: Speech corpora

This sections lists the source code of the experiment described in section 4.4.4, starting on page 148.

Listing B.35: runExperiment3.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2010-04-01
%%% run boundariness experiment 3: real audio data

addpath('functions');
outDirBase    = '/mount/corpora12/a2/duran_working/diss/segmentation/' ;
probeLengths  = 1:100;
lambda        = 2;
doPrintGraphs = true;
% speech signal encoding:
useMFCC  = true;
dataCols = [2 3 4 5 6 7 8 9 10 11 12 13];
% ---------------------------------------------------------------------------
% Curpus: IMS: SWMS
corpusDir = '/mount/corpora12/a2/duran_working/diss/db/IMSms/';
% segment label
%     2    d
%    80    d
%   147    d
%  1198    m
%  1772    m
%  3127    m
%  1903    QI
%  6902    QI
% 11014    QI
%    61    a:
%     4    s
%    88    Qu:
probeSegments = [2 147 1198 3127 1903 11014 61 4 88];
for pdx = 1:length(probeSegments),
    probeSegment = probeSegments(pdx);
    useRandomProbe = false;
    outDir = [outDirBase 'Exp3_IMS_s' num2str(probeSegment) '/'];
    experiment3( outDir, corpusDir, useMFCC, dataCols, probeSegment, probeLengths, ↩
       doPrintGraphs, useRandomProbe, lambda  );
    if pdx < 3,% do some tests with random probes
        useRandomProbe = true;
        outDir = [outDirBase 'Exp3_IMS_rnd/'];
        experiment3( outDir, corpusDir, useMFCC, dataCols, probeSegment, ↩
           probeLengths, doPrintGraphs, useRandomProbe, lambda  );
    end
end
% ---------------------------------------------------------------------------
```

```
% Curpus: PL-BOSS (2006A)
corpusDir = '/mount/corpora12/a2/duran_working/diss/db/2006A/';
% segment label
%    2     e
%    6     a
%    7     r
%    1     #t
% 3673     #t
%   95     #m
%  295     #d
%   62     #b
%  150     #b
%  343     #b
%  109     #m
%  225     #m
% 1219     #b
% 3759     #t
probeSegments = [1219 3759];%[150 343 109 225];%[95 295 62]; %[2 6 7 1 3673];
for pdx = 1:length(probeSegments),
    probeSegment = probeSegments(pdx);
    useRandomProbe = false;
    outDir = [outDirBase 'Exp3_PL_s' num2str(probeSegment) '/'];
    experiment3( outDir, corpusDir, useMFCC, dataCols, probeSegment, probeLengths, ↩
       doPrintGraphs, useRandomProbe, lambda  );
    if pdx < 3,
        useRandomProbe = true;
        outDir = [outDirBase 'Exp3_PL_rnd/'];
        experiment3( outDir, corpusDir, useMFCC, dataCols, probeSegment, ↩
           probeLengths, doPrintGraphs, useRandomProbe, lambda  );
    end
end
disp('Experiment␣3:␣all␣done.');
```

Listing B.36: functions/experiment3.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2009-08-04
function experiment3( outBase, corpusDir, useMFCC, dataCols, ...
    probeSegment, probeLengths, doPrintGraphs, useRandomProbe, lambda  )

% ---------------------------------------------------------------------------
% Load global settings and initialize the simulation environment:
bndSettings;
timestamp = datestr(now,'yyyy-mm-dd+HH-MM-SS');
outDir    = [outBase timestamp '/'];
if exist(outDir, 'dir'),
    sfx =1;
    od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    while exist(od, 'dir'),
```

```
        sfx = sfx+1;
        od = [outDir(1:end-1) '_' num2str(sfx) '/'];
    end
    outDir = od;
end
mkdir(outDir);
% Initialize log file and start logging:
prefix        = 'Exp3_' ;
diaryFilename = [outDir, prefix, timestamp, '.diary'];
diary(diaryFilename);
fprintf('[%s] start EXPERIMENT 3...\n', datestr(now,31) );
fprintf('                    Saving diary to %s\n', diaryFilename);
fprintf('                    Output base directory created in: %s\n', outDir);
fprintf('                    OMP_NUM_THREADS = %d\n', getenv('OMP_NUM_THREADS'));
% ------------------------------------------------------------------------------
if ~exist(corpusDir, 'dir'),
    error(['Input directory not found: ', corpusDir]);
else
    fprintf('Loading corpus data from: %s\n', corpusDir);
end
% Load signal data:
if useMFCC,
    sigFile = [corpusDir 'corpusMFCC.mat'];
    if ~exist(sigFile, 'file'),
        error(['Input data not found: ', sigFile]);
    else
        load([corpusDir 'corpusInfo.mat']);
        load([corpusDir 'segmentsPHON.mat']);
        load(sigFile);
        % create memory sequence for this experiment:
        MS = corpusMFCC.f(:,dataCols);
        n = find(isnan(MS));
        if n, % there are NaN values in MS, replace them with some near-zero random↩
            frames:
            disp('Replacing NaN frames in memory sequence');
            r = randn(size(n)) * nanZeroSd;
            MS(n) = r;
        end
        % create frame labels:
        szMS = size(MS,1);
        labels = zeros(szMS,1);
        sptr = 1;
        for fx=1:szMS,
            if fx > segmentsPHON(sptr,2),
                % next segment:
                sptr = sptr + 1;
            end
            labels(fx) = segmentsPHON(sptr,3);
        end
```

```
63
64          clear corpusMFCC;
65      end
66  else
67      error('Only␣MFCC␣implemented');
68  end
69  % ---------------------------------------------------------------------------
70  % Print some information about the memory sequence corpus:
71  disp(['[', datestr(now,31), ']␣speech␣corpus␣initialized:␣']);
72  disp(['␣␣␣size=', num2str(size(MS,1)), 'x', num2str(size(MS,2))])
73  if useRandomProbe,
74      % draw random frames from the corpus and make a probe from them:
75      r = randi(szMS, probeLengths(end), 1);
76      loc = r(1);
77      probe = MS(r,:);
78      fprintf('[%s]␣Start␣computations␣with␣random␣probe...\n',  datestr(now,31) );
79  else
80      loc = segmentsPHON(probeSegment,1);
81      probe = MS(loc:(loc+probeLengths(end)-1),:);
82      fprintf('[%s]␣Start␣computations␣with␣probe␣@␣index␣%d:␣[%s...],␣segment␣nr.␣%d↩
        ...\n', ...
83          datestr(now,31), loc, corpusInfo.lexPHON{segmentsPHON(probeSegment,3)}, ↩
            probeSegment );
84  end
85  % ---------------------------------------------------------------------------
86  [totalSimilarity, totalSimilarityN, rawHits, normHits, rawStd, normStd, fullRawSim,↩
        fullNormSim] = ...
87      getSimilarity(MS, probeLengths, probe, lambda);
88  fprintf('[%s]␣...done\n', datestr(now,31));
89  % Print some stats about the results:
90  fullRawSimDelta  = diff(fullRawSim);
91  fullNormSimDelta = diff(fullNormSim);
92  fprintf('full␣raw␣similarity:␣␣␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        fullRawSim), max(fullRawSim), mean(fullRawSim),  std(fullRawSim) );
93  fprintf('full␣raw␣sim.␣delta:␣␣␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        fullRawSimDelta), max(fullRawSimDelta), mean(fullRawSimDelta),  std(↩
        fullRawSimDelta) );
94  fprintf('full␣norm.␣similarity:␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        fullNormSim), max(fullNormSim), mean(fullNormSim),  std(fullNormSim) );
95  fprintf('full␣norm.␣sim.␣delta:␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        fullNormSimDelta), max(fullNormSimDelta), mean(fullNormSimDelta),  std(↩
        fullNormSimDelta) );
96  totalSimilarityDelta = diff(totalSimilarity);
97  totalSimilarityNDelta = diff(totalSimilarityN);
98  fprintf('raw␣similarity:␣␣␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        totalSimilarity), max(totalSimilarity), mean(totalSimilarity),  std(↩
        totalSimilarity) );
99  fprintf('raw␣sim.␣delta:␣␣␣min=%8.3f;␣max=%8.3f;␣mean=%8.3f;␣sd=%8.3f\n', min(↩
        totalSimilarityDelta), max(totalSimilarityDelta), mean(totalSimilarityDelta),  ↩
```

```
    std(totalSimilarityDelta) );
fprintf('norm. similarity: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarityN), max(totalSimilarityN), mean(totalSimilarityN),  std(totalSimilarityN) );
fprintf('norm. sim. delta: min=%8.3f; max=%8.3f; mean=%8.3f; sd=%8.3f\n', min(totalSimilarityNDelta), max(totalSimilarityNDelta), mean(totalSimilarityNDelta), std(totalSimilarityNDelta) );
%%% %%% %%%
%%% store results:
ts=['_' num2str(probeSegment) '_'];
outFileName = [outDir, prefix,  ts, 'rawSim.txt'];
save(outFileName,'totalSimilarity', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir, prefix,  ts, 'normSim.txt'];
save(outFileName,'totalSimilarityN', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir, prefix,  ts, 'rawStd.txt'];
save(outFileName,'rawStd', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
outFileName = [outDir, prefix,  ts, 'normStd.txt'];
save(outFileName,'normStd', '-ASCII');
disp(['[', datestr(now,31), '] saved result to file ', outFileName ]);
%%% evaluate : print graphs
if doPrintGraphs,
    printAllGraphs( outDir, MS(loc:(loc+probeLengths(end)-1),:), totalSimilarity, totalSimilarityN, ...
        rawHits, rawStd, normHits, normStd, fullNormSim, fullRawSim, ...
        probeLengths, [timestamp ts], labels, corpusInfo.lexPHON, loc, prefix );
else
    disp(['[', datestr(now,31), '] (skipped printing graphics)']);
end
%%% END
disp(['[', datestr(now,31), '] all done. bye!']);
diary off;
```

B.2. EMA clustering

This section lists the source code for the clustering experiments presented in chapter 5.

B.2.1. Experiment 1

Listing B.37: clusteringExperiment1.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% based on: "exportForClustering.m"  [2010-12-07]
%%%           "clusteringEvaluation.m" [2010-12-07]
```

```
%%%            "code/csm_EMMA/exportForClusteringEq.m"          [2011-03-22]
%%%            "code/csm_EMMA/clusteringEvaluationCombinedEq.m" [2011-03-23]
%%% Reference: Duran et al. Interspeech, 2011

%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('evaluation');
addpath('../io');
THISMFILENAME = 'clusteringExperiment1';
% ==============================================================================
% Initialize experiment setup
% ------------------------------------------------------------------------------
setup = struct([]);
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/clustering/';
setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};
% The data types to use in the experiment: specified by a matrix, with
%  first column: articulatory data EMA;
%  second column: audio data ACC
%  -> (1) EMA (2) ACC (3) EMA+ACC
setup.dataTypes = [1 0 ; 0 1 ; 1 1];
%  labels used for output files:
setup.dataLabels = {'EMA', 'ACC', 'EMA+ACC'};
% Selecting a specific audio representation:
setup.useENVB   = true; % use amplitude envelope bands
setup.useMFCC   = false;% use MFCC
setup.useWAVE   = false;% (using the raw wave signal is not supported)
% Delta1 and Delta2 are the signals' first and second derivatives:
setup.useDelta1 = true;
setup.useDelta2 = true;
% ------------------------------------------------------------------------------
% Function handle for the function which reads the data from the
% pre-processed corpus files and create the data structures used in the
% clustering experiment:
setup.dataImportFH = @getClusteringDataFromPolishEMA;
% ------------------------------------------------------------------------------
% Function handle for the function which takes a sample set from the
% corpus:
setup.samplingFH = @getSampleSet;
% ------------------------------------------------------------------------------
% Set how many samples per class should be exported as a ratio of the smallest
% phone class for a speaker, e.g. 0.75 means that the exported classes will all
% be 75% of the size of the smallest phone class
setup.minClusterRatio = 0.75;
% Set the number of repeated runs of the clustering procedure on a sample set:
setup.RUNS = 1000;
% ------------------------------------------------------------------------------
```

```
% Set handle to a clustering function:
setup.clusterFH = @bisectingKmeans;
% The following parameters are relevant only for bisecting k-means
% clustering:
% Number of iterations:
setup.BKMiter = 10;
% Use cluster refinement: {true,false}
setup.BKMrefine = true;
% ---------------------------------------------------------------------------
% Define sets of classes to evaluate confusions between "broad phonetic
% classes":
setup.broadClasses = {[1 2], [3 4], [5 6 7], 8 };
setup.broadClassLabels = {'K', 'R', 'Y', 'A'};
% ---------------------------------------------------------------------------
% Evaluation
% set path and file name for the printing R script:
% (MATLAB cannot embedd fonts into PDF files)
setup.RSCRIPT  = '../../r/printConfusionMatrix.R';
setup.RSCRIPT2 = '../../r/printConfusionMatrix2.R';
% ---------------------------------------------------------------------------
% print debugging info to console (and log file)
setup.VERBOSE=true;
% ---------------------------------------------------------------------------
% Set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, setup.timestamp, '/' ] ;
% Save terminal output to a log file
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
% ===========================================================================
%% run %%%
% Start logging and print experiment setup:
diary(diaryFilename);
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp(['pwd␣=␣', pwd]);% current working directory
    disp( '␣␣Setup:␣');
    disp(setup);
end
% This runs the actual experiment:
experiment1(setup);
%% finished
disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣', ↩
   setup.timestamp, '.␣Bye!']);
diary off;% close log file
```

Listing B.38: functions/experiment1.m

```
function experiment1( setup )
%EXPERIMENT1 -- Run clustering experiment on EMA data
%
% Run clustering experiment on segmental acoustic and articulatory speech
% data.
% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart

% Open Matlab worker pool to enable running parallel computations in "parfor" loop:
matlabpool ;

% rows:
%  clustering runs
% columns:
%  (:,1)  -- speakerNr
%  (:,2)  -- dT: data type number
%  (:,3)  -- nn: iteration number
%  (:,4)  -- clustering OK flag
%  (:,5)  -- Purity
%  (:,6)  -- V-measure
%  (:,7)  -- J_pur  clusterClasses
%  (:,8)  -- J_v    clusterClasses
%  (:,9)  -- caM    classification error on matched table
%  (:,10) -- caS    classification error on sorted table
%  (:,11) -- ari
%  (:,12) -- Jari
sampleSets = NaN(size(setup.dataTypes,1) * size(setup.speakerDirs,1) * setup.RUNS, ↩
    12);

ptrSample = 1;

% Loop through all speaker sub-corpora:
for speakerNr = 1:length(setup.speakerDirs),
    % Working parameters (autosettings):
    speaker =  setup.speakerDirs{speakerNr}(1:end-1) ;
    fprintf('\n==␣Start␣speaker␣%s==\n', speaker);
    % ==========================================================================
    % == Initializations for current speaker and data type ==
    load([setup.matlabDir setup.speakerDirs{speakerNr} 'index.mat'],'corpusFs');
    fprintf('␣␣corpusFs=%dHz\n', corpusFs);
    % Collect file names with the confusion tables for this speaker (to
    % produce graphs at the end of the experiment):
    csvFileNamesMatched = cell(size(setup.dataTypes,1),1);
    csvFileNamesSorted = cell(size(setup.dataTypes,1),1);
    % --------------------------------------------------------------------------
    % Loop through all data type conditions:
    for dT = 1:size(setup.dataTypes,1),
        useART = setup.dataTypes(dT,1);% use articulatory signal data?
        useACC = setup.dataTypes(dT,2);% use acoustic signal data?
        fprintf('=␣Data␣type:␣%s␣=\n', setup.dataLabels{dT} );
```

```
49          % Import the required data from the corpus for the current
50          % parameter settings:
51          [D, IDX, lex] = setup.dataImportFH(setup, speakerNr, useACC, useART);
52          rows     = length(IDX);% The total number of data frames
53          numClass = length(lex);% The total number of reference classes
54          % Compute class label frequencies in corpus and print them:
55          classFrq = zeros(numClass,1);
56          for fx = 1:rows,
57              p = IDX(fx);
58              classFrq(p) = classFrq(p) + 1;
59          end
60          fprintf('Label␣frequencies␣(lex.idx):\n');
61          for c = 1:numClass,
62              fprintf('␣␣%3s:␣%5d␣␣(%2d)\n', lex{c}, classFrq(c), c);
63          end
64          % Determine the target size for clusters such that each class set is
65          % of equal size:
66          clusterSz = round( min(classFrq) * setup.minClusterRatio ) ;
67          fprintf('␣>>␣Using␣%d␣items␣per␣class␣for␣%s\n', clusterSz, speaker);
68          % Initialize containers to collect the results of the individual
69          % clusterings -> 3-dimensional matrix with:
70          %   rows:    clusters
71          %   columns: classes
72          %   levels:  iterations
73          allConfusionsMatched = NaN(numClass, numClass, setup.RUNS);
74          allConfusionsSorted = NaN(numClass, numClass, setup.RUNS);
75          allConfusionSortedIdx = NaN(numClass,1,setup.RUNS);
76          sampleNumbers = NaN(setup.RUNS, 1);
77          % Count some statistics about lost rows in confusion tables:
78          lostRows = NaN(setup.RUNS, 1);
79          ari_c  = sum(arrayfun(@(x) nchoosek(x,2), repmat(clusterSz,1,numClass)));
80          ari_n2 = nchoosek(clusterSz*numClass,2);
81          % ---------------------------------------------------------------------
82          % Repeat clustering on random sample sets from corpus.
83          % Run parallel for-loop:
84          parfor it = 1:setup.RUNS,
85              % Create sample with equally represented classes:
86              [Deq,IDXeq] = setup.samplingFH(setup,D,IDX,numClass,clusterSz);
87              % ===================================================================
88              % Do clustering
89              % Initialize variables:
90              IDXhyp = [];
91              ok     = true;
92              pur    = NaN;% Purity
93              v      = NaN;% V-measure
94              Jpur   = NaN;
95              Jv     = NaN;
96              caM    = NaN;% classification error (matched)
97              caS    = NaN;% classification error (sorted)
```

```
            ari    = NaN;
            Jari   = NaN;
            try
                [ IDXhyp ] = setup.clusterFH( Deq, numClass, setup );
            catch err
                ok = false;
                warning(['Bisecting␣k-means␣with␣refinement␣failed␣for␣'...
                    speaker ';␣dT=' num2str(dT), ';␣iteration=' num2str(it)]);
                disp(err);
            end
            numClust = length(unique(IDXhyp));% Number of clusters
            if numClust ~= numClass,
                fprintf('WARNING:␣number␣of␣clusters␣(%d)␣does␣not␣match␣number␣of␣↩
                   classes␣(%d)\n', numClust, numClass);
                ok = false; % ?
            end
            if ok,
                % ==============================================================
                % Evaluate clustering results
                % --------------------------------------------------------------
                % Assign clusters to reference classes:
                %   rows:    clusters
                %   columns: classes
                % Each cell (i,j) contains the total number of frames of a
                % cluster i that belong to the reference class j
                % -> This is not yet a cluster-classes confusion matrix!
                %    The rows and columns are not ordered according to their
                %    mutual correspondences.
                clusterClasses = assignClassesToClusters( IDXhyp, IDXeq);
                if sum(sum(clusterClasses)) ~= (clusterSz * numClass),
                    error(['Number␣of␣items␣(',num2str(sum(sum(clusterClasses)))↩
                       ,...
                        ')␣not␣as␣expected␣(',num2str(clusterSz * numClass),')␣in␣↩
                           clusterClasses']);
                end
                % Compute evaluation measures:
                pur = purity( clusterClasses );
                v   = vmeasure( clusterClasses );
                ari = adjustedRandIndex( clusterClasses, ari_c, ari_n2 );
                % Apply Andrew Rosenberg's ClusterEvaluator
                % See: http://eniac.cs.qc.cuny.edu/andrew/
                % Just for debugging: This here is a wrapper that
                % internally calles the actual Java methods:
                [Jpur Jv Jari] = rosenbergEval(clusterClasses) ;
                % --------------------------------------------------------------
                % compute matched confusion matrix
                thisConfusionMatched = zeros(numClass,numClass);
                for c = 1:numClust,
                    cluster = clusterClasses(c,:);
```

```
                    [maxVal maxIndx] = max(cluster);
                    thisConfusionMatched(maxIndx,:) = thisConfusionMatched(maxIndx,:) + cluster;
                end
                if sum(sum(thisConfusionMatched)) ~= (clusterSz * numClass),
                    disp(clusterClasses);
                    disp(thisConfusionMatched);
                    error(['Number of items (',num2str(sum(sum(thisConfusionMatched))),...
                        ') not as expected (',num2str(clusterSz * numClass),') in thisConfusion']);
                end
                lostRows(it) = numClass - length(find(sum(thisConfusionMatched,2)));
                % ------------------------------------------------------------
                % compute sorted confusion matrix
                [ccVal ccIdx] = sort(clusterClasses,2,'descend');
                % primary key:   column 1 in ccIdx
                % secondary key: column 2 in ccIdx... etc
                keys = zeros(size(ccIdx));
                keys(:,1) = ccIdx(:,1);
                for c = 2:numClust,
                    keys(:,c) = abs(ccIdx(:,1) - ccIdx(:,c));
                end
                [kyVal kyIdx] = sortrows(keys);
                thisConfusionSorted = clusterClasses(kyIdx,:);
                % ------------------------------------------------------------
                % Classification error
                caM = classificationAccuracy(thisConfusionMatched);
                caS = classificationAccuracy(thisConfusionSorted);
                % ------------------------------------------------------------
                % clusterClasses ccIdx keys kyIdx
                allConfusionsMatched(:,:,it) = thisConfusionMatched ;
                allConfusionsSorted(:,:,it) = thisConfusionSorted ;
                allConfusionSortedIdx(:,:,it) = ccIdx(:,1);% the class index for each cluster in the sorted tables
                sampleNumbers(it) = ptrSample + it - 1;

            end % if ok (clustering without errors)
            localSampleSets(it,:) = [speakerNr dT it ok pur v Jpur Jv caM caS ari Jari];
        end  % for it
        if size(localSampleSets,1) ~= setup.RUNS,
            error('Number of iterations not as specified!');
        end
        thisRange = ptrSample:(ptrSample+setup.RUNS-1);
        sampleSets(thisRange,:) = localSampleSets;
        ptrSample = ptrSample + setup.RUNS;
        % ======================================================================
```

```
187         % Evaluate and print results
188         % ----------------------------------------------------------------------
189         avgPurity = sum(sampleSets(thisRange,5)) / length(thisRange);
190         avgV      = sum(sampleSets(thisRange,6)) / length(thisRange);
191         stdPur = std(sampleSets(thisRange,5));
192         stdV   = std(sampleSets(thisRange,6));
193         avgJPurity = sum(sampleSets(thisRange,7)) / length(thisRange);
194         avgJV      = sum(sampleSets(thisRange,8)) / length(thisRange);
195         stdJPur = std(sampleSets(thisRange,7));
196         stdJV   = std(sampleSets(thisRange,8));
197         avgCaM = sum(localSampleSets(:,9)) / setup.RUNS;
198         avgCaS = sum(localSampleSets(:,10)) / setup.RUNS;
199         stdCaM = std(localSampleSets(:,9));
200         stdCaS = std(localSampleSets(:,10));
201         fprintf('== RESULTS ==\n  Speaker: %s\n', speaker )
202         fprintf('  Purity:\n')
203         fprintf('    average: %5.3f (%8.6f)\n', avgPurity, avgPurity);
204         fprintf('    std:     %5.3f (%8.6f)\n', stdPur, stdPur );
205         fprintf('  V-measure:\n')
206         fprintf('    average: %5.3f (%8.6f)\n', avgV, avgV);
207         fprintf('    std:     %5.3f (%8.6f)\n', stdV, stdV );
208         fprintf('  Purity (J):\n')
209         fprintf('    average: %5.3f (%8.6f)\n', avgJPurity, avgJPurity);
210         fprintf('    std:     %5.3f (%8.6f)\n', stdJPur, stdJPur );
211         fprintf('  V-measure (J):\n')
212         fprintf('    average: %5.3f (%8.6f)\n', avgJV, avgJV);
213         fprintf('    std:     %5.3f (%8.6f)\n', stdJV, stdJV );
214         fprintf('  Classification accuracy ("matched"):\n')
215         fprintf('    average: %5.3f (%8.6f)\n', avgCaM, avgCaM);
216         fprintf('    std:     %5.3f (%8.6f)\n', stdCaM, stdCaM );
217         fprintf('  Classification accuracy ("sorted"):\n')
218         fprintf('    average: %5.3f (%8.6f)\n', avgCaS, avgCaS);
219         fprintf('    std:     %5.3f (%8.6f)\n', stdCaS, stdCaS );
220         avgAri  = sum(localSampleSets(:,11)) / setup.RUNS;
221         avgJari = sum(localSampleSets(:,12)) / setup.RUNS;
222         stdAri  = std(localSampleSets(:,11));
223         stdJari = std(localSampleSets(:,12));
224         fprintf('  Adjusted Rand index:\n')
225         fprintf('    average: %5.3f (%8.6f)\n', avgAri, avgAri);
226         fprintf('    std:     %5.3f (%8.6f)\n', stdAri, stdAri );
227         fprintf('  Adjusted Rand index (J):\n')
228         fprintf('    average: %5.3f (%8.6f)\n', avgJari, avgJari);
229         fprintf('    std:     %5.3f (%8.6f)\n', stdJari, stdJari );
230         % ----------------------------------------------------------------------
231         % Matched confusion tables:
232         csvFileNamesMatched{dT} = confusionEvaluation(setup, allConfusionsMatched, ↩
               [], 'Matched', speaker, dT, lex, clusterSz );
233         % ----------------------------------------------------------------------
234         % Matched confusion tables:
```

```
235         csvFileNamesSorted{dT} = confusionEvaluation(setup, allConfusionsSorted, ↩
               allConfusionSortedIdx,'Sorted', speaker, dT, lex, clusterSz );
236         % ------------------------------------------------------------------
237         % Save individual results to mat file for later inspection:
238         matFile = [setup.baseDir speaker '_' setup.dataLabels{dT} '_results.mat'];
239         save(matFile, 'allConfusionsMatched', 'allConfusionsSorted', 'sampleNumbers↩
               ', 'localSampleSets', 'lostRows' );
240         fprintf('  Stored results matrices to %s\n\n', matFile);
241     end % for dT
242     % ---------------------------------------------------------------------
243     % Matched confusions:
244     outFilePDFrel = [setup.baseDir speaker '_confusion_AllM_rel.pdf'] ;
245     outFilePDFabs = [setup.baseDir speaker '_confusion_AllM_abs.pdf'] ;
246     [result status] = r(setup.RSCRIPT, csvFileNamesMatched{1}, csvFileNamesMatched↩
           {2}, csvFileNamesMatched{3}, outFilePDFrel, outFilePDFabs);
247     if status ~= 0,
248         warning(['R returned status code: ' num2str(status)]);
249         disp(['[', datestr(now,31), '] >>> Output of R script:']);
250         disp(result);
251         disp(' <<< END script output.');
252     else
253         disp(['[', datestr(now,31), '] Saved matched confusion matrix to PDF: ', ↩
               outFilePDFrel ]);
254     end
255     % print graph of confusion matrix to PDF (EMA and ENV only):
256     outFilePDFrel2 = [setup.baseDir speaker '_confusion_EEM_rel.pdf'] ;
257     outFilePDFabs2 = [setup.baseDir speaker '_confusion_EEM_abs.pdf'] ;
258     [result status] = r(setup.RSCRIPT2, csvFileNamesMatched{1}, csvFileNamesMatched↩
           {2}, outFilePDFrel2, outFilePDFabs2);
259     if status ~= 0,
260         warning(['R returned status code: ' num2str(status)]);
261         disp(['[', datestr(now,31), '] >>> Output of R script:']);
262         disp(result);
263         disp(' <<< END script output.');
264     else
265         disp(['[', datestr(now,31), '] Saved matched confusion matrix to PDF: ', ↩
               outFilePDFrel2 ]);
266     end
267     % ---------------------------------------------------------------------
268     % Matched confusions:
269     outFilePDFrel = [setup.baseDir speaker '_confusion_AllS_rel.pdf'] ;
270     outFilePDFabs = [setup.baseDir speaker '_confusion_AllS_abs.pdf'] ;
271     [result status] = r(setup.RSCRIPT, csvFileNamesSorted{1}, csvFileNamesSorted↩
           {2}, csvFileNamesSorted{3}, outFilePDFrel, outFilePDFabs);
272     if status ~= 0,
273         warning(['R returned status code: ' num2str(status)]);
274         disp(['[', datestr(now,31), '] >>> Output of R script:']);
275         disp(result);
276         disp(' <<< END script output.');
```

```
    else
        disp(['[', datestr(now,31), '] Saved sorted confusion matrix to PDF: ', outFilePDFrel ]);
    end
    % print graph of confusion matrix to PDF (EMA and ENV only):
    outFilePDFrel2 = [setup.baseDir speaker '_confusion_EES_rel.pdf'] ;
    outFilePDFabs2 = [setup.baseDir speaker '_confusion_EES_abs.pdf'] ;
    [result status] = r(setup.RSCRIPT2, csvFileNamesSorted{1}, csvFileNamesSorted{2}, outFilePDFrel2, outFilePDFabs2);
    if status ~= 0,
        warning(['R returned status code: ' num2str(status)]);
        disp(['[', datestr(now,31), '] >>> Output of R script:']);
        disp(result);
        disp(' <<< END script output.');
    else
        disp(['[', datestr(now,31), '] Saved sorted confusion matrix to PDF: ', outFilePDFrel2 ]);
    end
end % for speakerNr
% Save individual results to mat file for later inspection:
matFile = [setup.baseDir 'resultSets.mat'];
save(matFile, 'sampleSets' );
fprintf('  Stored results overview to %s\n\n', matFile);
fprintf('%,5d of %,5d runs OK\n', find( sampleSets(:,4) ), size(sampleSets,1));
fprintf('Results stored to: %s\n', setup.baseDir);
matlabpool close;
end
```

Listing B.39: functions/getClusteringDataFromPolishEMA.m

```
function [ D, IDX, lex ] = getClusteringDataFromPolishEMA( setup, speakerNr, useACC, useART )
%
%  D   = (N x d)  data set of N d-dimensional data points
%
%  IDX = (N x 1)  class index labels for each data point (reference
%                 classification used for external evaluation)
%
%  lex = {C x 1}  cell array with string labels for each of the C classes

% ------------------------------------------------------------------------
% load index and label data for this speaker:
inDir = [setup.matlabDir setup.speakerDirs{speakerNr}];
load([inDir 'index.mat']);

% check whether the variables are loaded successfully
if ~exist('baseNames','var') || ...
        ~exist('totalFramesEma','var') || ~exist('totalFramesWav','var')  || ...
        ~exist('totalFramesEnv','var') || ~exist('totalFramesMFCC','var') || ...
```

```
19          ~exist('segmentsPHON','var')   || ~exist('segmentsWORD','var') || ...
20          ~exist('corpusFs','var'),
21      % add more checks?
22      error('index.mat␣was␣not␣opened␣successfully!');
23  end
24  % ----------------------------------------------------------------------
25  %%
26  % Determine the number of columns for the output data structurs (each
27  % column corresponds to one "dimension" of the data):
28  offset=0;
29  accCols = [];
30  emaCols = [];
31  envCols = [];
32  mfcCols = [];
33  if useACC,
34      if setup.useWAVE,
35          disp('WARNING:␣adding␣WAVE␣not␣implemented!');
36      end
37      if setup.useMFCC,
38          plus = 13;
39          if setup.useDelta1 && setup.useDelta2,
40              plus=39;
41          else
42              if setup.useDelta1 || setup.useDelta2,
43                  plus=26;
44              end
45          end
46          mfcCols = (offset+1) : (offset+plus);% the MFCC columns in "corpus"
47          offset = offset + plus;
48      end
49      if setup.useENVB,
50          % use amplitude envelop representation (default for audio)
51          plus = 8;
52          if setup.useDelta1 && setup.useDelta2,
53              plus=24;
54          else
55              if setup.useDelta1 || setup.useDelta2,
56                  plus=16;
57              end
58          end
59          envCols = (offset+1) : (offset+plus);% the envelop columns in "corpus"
60          offset = offset + plus;
61      end
62
63      accCols = [mfcCols envCols];
64  end
65  if useART,
66      plus = 8;
67      if setup.useDelta1 && setup.useDelta2,
```

```
        plus=24;
    else
        if setup.useDelta1 || setup.useDelta2,
            plus=16;
        end
    end
    emaCols = (offset+1) : (offset+plus);
    offset = offset + plus;
end
%%
% ---------------------------------------------------------------------------
% initialize output data
rows   = max([totalFramesEma, totalFramesEnv, totalFramesMFCC]) ;
D   = zeros(rows, offset);
IDX = zeros(rows,1);
ptr    = 1 ;
ptrEnd = 0;
% loop through utterances (files)
for fdx=1:length(baseNames), %5, %
    bfile = baseNames{fdx};
    skip  = false;
    if useART,
        if sum(checkFiles(fdx,1:4)) ~= 4,
            if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣EMA␣↩
               data']); end
            skip = true ;
        end
    end
    if useACC || setup.useWAVE || setup.useMFCC,
        if checkFiles(fdx,5) ~= 1,
            if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣Audio␣↩
               data']); end
            skip = true ;
        end
    end
    if skip,
        continue;
    end
    % the segments of this utterance:
    seg = segmentsPHON( segmentsPHON(:,4)==fdx , :);
    %  segmentsPHON = (S_s x 6) matrix, where S_s is the number of
    %  resulting segments of the annotated signal s
    %    column 1: the start index of the current segment
    %    column 2: the end index of the current segment
    %    column 3: the label type number in lex* corresponding to the
    %              segment (or 0 for unlabeled parts)
    %    column 4: the index of the corresponding file in baseNames
    %    column 5: local start time of segment in file
    %    column 6: local end time of segment in file
```

```
115     if length(find( seg(:,3) )) ~= size(seg,1) ,
116         disp(['Warning:␣unlabeled␣segment␣found␣in␣' baseNames{fdx}, '␣(fdx=', ↩
               num2str(fdx), ')␣-->␣skipping!']);
117         continue;
118         % else
119         %    disp(['No unlabeled segment found in ' baseNames{fdx}, ' (fdx=', ↩
               num2str(fdx), ')'])
120     end
121     dataART = [];
122     dataACC = [];
123     idxACC = [];
124     idxART = [];
125     % == EMA ======================================================== EMA ==
126     if useART,
127         % Add articulatory data from corpus
128         % load EMA data for the current utterance:
129         load([inDir bfile '_ema.mat']);
130         % reads from file:
131         %   signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y,
132         %   signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy
133         load([inDir bfile '_frames.mat']);
134         % reads from file:
135         %   frames  -- the labels per frame for each utterance
136         %            = (N x 5), where the columns correspond to the
137         %            labels from:
138         %                 1 -- .words
139         %                 2 -- .cv
140         %                 3 -- .tbo
141         %                 4 -- .tt
142         %                 5 -- .eucl
143         from = seg(1,1);
144         to   = seg(end,2);
145         [dataART from to] = m_getChunk(from, to, setup.VERBOSE, setup.useDelta1, ↩
               setup.useDelta2, ...
146             signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
147             signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy );
148         idxART = frames(from:to,2);
149     end % if useART
150     % == ENV ======================================================== ENV ==
151     if useACC,
152         % load ENV data for the current utterance:
153         load([inDir bfile '_audio.mat']);
154         load([inDir bfile '_frames.mat']);
155         from = seg(1,1);
156         to   = seg(end,2);
157         if setup.useMFCC,
158             [lcM from to] = m_getMFCCChunk(from, to, signalMFCC, ...
159                 setup.useDelta1, setup.useDelta2, setup.VERBOSE );
160         else
```

```
            lcM=[];
        end
        if setup.useENVB,
            [lcE from to] = m_getEnvChunk(from, to, signalENVB, ...
                setup.useDelta1, setup.useDelta2, setup.VERBOSE );
        else
            lcE=[];
        end
        dataACC = [lcM lcE];

        idxACC = frames(from:to,2);
    end % if useACC
    if useACC && useART,
        if (length(dataACC) ~= length(dataART)),
            disp('Warning: data lengths do not match!');
            ptrEnd = ptr + min([length(dataACC), length(dataART)]) - 1;
        else
            ptrEnd = ptr + length(dataACC) - 1;
        end
        if ~isempty(find(idxART~=idxACC, 1)),
            disp('Warning: labels mismatch. Skipping!');
            skip=true;
        end
    else
        % dataACC or dataART is empty
        ptrEnd = ptr + max([length(dataACC), length(dataART)]) - 1;
    end
    if skip,
        continue;
    end
    if useACC,
        D(ptr:ptrEnd,accCols) = dataACC;
        IDX(ptr:ptrEnd) = idxACC;
    end
    if useART,
        D(ptr:ptrEnd,emaCols) = dataART;
        IDX(ptr:ptrEnd) = idxART;
    end
    ptr = ptrEnd + 1;

end % for fdx
if ptrEnd < rows,
    % truncate output data to actual length
    D   = D(1:ptrEnd,:);
    IDX = IDX(1:ptrEnd);
end
lex = lexPHON;
if setup.VERBOSE,
    fprintf('[%s %s] Finished data import. Total number of samples: %d\n', ...
```

```
        datestr(now,31), mfilename, ptrEnd );
end
end % getClusteringDataFromPolishEMA

% ==========================================================================
%% get chunk from the signal in the range [from:to]
% added 2012-08-28: use both x and y EMA signals
function [out from to] = m_getChunk(from, to, VERBOSE, useDelta1, useDelta2, ...
    signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
    signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE', ['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalLLIPx),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalLLIPx)), '!' ]);
    end
    to = length(signalLLIPx);
end
if useDelta1 && useDelta2,
    dims = 24;
else
    if useDelta1 || useDelta2,
        dims = 16;
    else
        dims = 8;
    end
end
out = zeros(to - from +1, dims);
out(:,1) = signalLLIPx(from:to,1);
out(:,2) = signalLLIPy(from:to,1);
out(:,3) = signalTBO1x(from:to,1);
out(:,4) = signalTBO1y(from:to,1);
out(:,5) = signalTBO2x(from:to,1);
out(:,6) = signalTBO2y(from:to,1);
out(:,7) = signalTTIPx(from:to,1);
out(:,8) = signalTTIPy(from:to,1);
if useDelta1,
    out(:,9)  = signalLLIPx(from:to,2);
    out(:,10) = signalLLIPy(from:to,2);
    out(:,11) = signalTBO1x(from:to,2);
    out(:,12) = signalTBO1y(from:to,2);
    out(:,13) = signalTBO2x(from:to,2);
```

```
    out(:,14) = signalTBO2y(from:to,2);
    out(:,15) = signalTTIPx(from:to,2);
    out(:,16) = signalTTIPy(from:to,2);
end
if useDelta2,
    out(:,17) = signalLLIPx(from:to,3);
    out(:,18) = signalLLIPy(from:to,3);
    out(:,19) = signalTBO1x(from:to,3);
    out(:,20) = signalTBO1y(from:to,3);
    out(:,21) = signalTBO2x(from:to,3);
    out(:,22) = signalTBO2y(from:to,3);
    out(:,23) = signalTTIPx(from:to,3);
    out(:,24) = signalTTIPy(from:to,3);
end
end

% ======================================================================
%%
function [out from to] = m_getEnvChunk(from, to, signalENVB, useDelta1, useDelta2, ↩
   VERBOSE )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing invalid start index from ', ...
            num2str(from), ' to 1!']);
    end
    from = 1;
end
if to > length(signalENVB),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing invalid end index from ', ...
            num2str(to), ' to ' num2str(length(signalENVB)), '!' ]);
    end
    to = length(signalENVB);
end
if useDelta1 && useDelta2,
    out = signalENVB(from:to, :);
else
    if useDelta1,
        out = signalENVB(from:to, 1:16);
    else
        if useDelta2,
            out = signalENVB(from:to, [1:8 17:24]);
        else
            out = signalENVB(from:to, 1:8);
        end
    end
end
end
```

```

% ==========================================================================
%%
function [out from to] = m_getMFCCChunk(from, to, signalMFCC, useDelta1, useDelta2,↩
    VERBOSE )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalMFCC),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalMFCC)), '!' ]);
    end
    to = length(signalMFCC);
end
if useDelta1 && useDelta2,
    out = signalMFCC(from:to, :);
else
    if useDelta1,
        out = signalMFCC(from:to, 1:16);
    else
        if useDelta2,
            out = signalMFCC(from:to, [1:8 17:24]);
        else
            out = signalMFCC(from:to, 1:8);
        end
    end
end
end
```

Listing B.40: functions/getSampleSet.m

```
function [ Deq, IDXeq ] = getSampleSet(setup, D, IDX, numClass, clusterSz)

rows  = length(IDX);
% Define a helper variable
indicesEq = zeros(rows, numClass);
for c = 1:numClass,
    % assuming that the class indices form an incremental list starting at 1,
    % the loop index c is equal to the class label
    % Get all frame indices from the current phone class:
    currentClass = find( IDX == c) ;
    % Shuffle the indices:
    sx = shuffle(currentClass);
```

```
    % ....and take the first <clusterSz> frames from the list:
    indicesEq(sx(1:clusterSz), c) = 1;
end
% Assign the original frames from D to the output according to the shuffled indices↩
    :
indices = find( sum(indicesEq,2) );
Deq   = D(indices,:);
IDXeq = IDX(indices);
end
```

Listing B.41: functions/shuffle.m

```
function [ out ] = shuffle( vec )
% [ out ] = shuffle( vec ) -- shuffles the rows in vec
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-03-21
m   = size(vec,1);
out = NaN(size(vec)) ;
chk = zeros(m,1);
for x = 1:m-1,
    rep = true;
    while rep,
        rx = randi(m);
        if chk(rx) == 0,
            out(rx,:) = vec(x,:);
            chk(rx) = 1;
            rep = false;
        else
            continue;
        end
    end
end
out( chk==0, : ) = vec(m,:);
end
```

Listing B.42: functions/bisectingKmeans.m

```
function [ C ] = bisectingKmeans( D, k, setup )
%BISECTINGKMEANS Bisecting k-means algorithm for finding k clusters
%
%  INPUT:
%    D            -- (n x d) matrix with n d-dimensional samples
%    k            -- the desired number of clusters
%   *iter         -- the number of iterations for the bisecting step
%   *doRefinement --
%
%  OUTPUT:
%    C    -- (n x 1) array containing the cluster ID for each sample
%
```

```
13 %  Basic Bisecting K-means Algorithm
14 %  ----------------------------------------------
15 %  Steinbach, M.; Karypis, G. & Kumar, V.
16 %  A comparison of document clustering techniques
17 %  KDD Workshop on Text Mining, 2000
18 %  -- or --
19 %  Steinbach, M.; Karypis, G. & Kumar, V.
20 %  A Comparison of Document Clustering Techniques
21 %  Techreport University of Minnesota -
22 %  Computer Science and Engineering, 2000
23 %  ----------------------------------------------
24 %  1. Pick a cluster to split
25 %  2. Find two sub-clusters using basic K-means
26 %  3. Repeat (2) for ITER times and take the split
27 %     that produces the clustering with highest
28 %     overall similarity
29 %  4. Repeat (1-2-3) until the desired number k of
30 %     clusters is reached
31 %  ----------------------------------------------
32 %
33 %  Note:
34 %  The k-means algorithm used by Steinbach et al. is
35 %  incrementally updating the centroids after each
36 %  assignment of a data point.
37
38 if nargin < 2,
39     error('Missing␣required␣input␣arguments:␣D,␣k␣');
40 end
41 if nargin == 3,
42     iter         = setup.BKMiter;
43     doRefinement = setup.BKMrefine;
44 else
45     iter = 5;
46     doRefinement = 0;
47 end
48 [N dim] = size(D);
49 C = ones(N,1);
50 % check k
51 if 1 > k || k > N,
52     error('Parameter␣k␣must␣be␣in␣range␣[1,N]');
53 end
54 clustersFound = 1; % trivial cluster containing all data samples
55 clusterCentroids = NaN(k,dim);
56 while ( clustersFound < k ),
57     % determine largest cluster
58     maxCID = 0;
59     maxCSZ = 0;
60     for cx=1:k,
61         clust = find(C==cx);
```

```
        csz = size(clust,1);
        if (csz > maxCSZ),
            maxCID = cx;
            maxCSZ = csz;
        end
    end
    % split the largest remaining cluster
    DsplitIDX = find(C == maxCID);
    Dsplit    = D(DsplitIDX,:);
    bestSplit = zeros(size(Dsplit,1),1);
    bestCent  = [];
    maxOS = -1;
    for it = 1:iter,
        [IDX CENT] = kmeans(Dsplit,2);
        % determine overall similarity of the current solution
        thisOS = overallSimilarity(Dsplit, IDX, CENT);
        if ( thisOS > maxOS ),
            maxOS = thisOS;
            bestSplit = IDX;
            bestCent  = CENT;
        end
    end
    % increment cluster counter and repeat
    clustersFound = clustersFound +1;
    newLabels = zeros(size(bestSplit));
    newLabels( bestSplit==1 ) = maxCID;
    newLabels( bestSplit==2 ) = clustersFound;
    clusterCentroids(maxCID,:) = bestCent(1,:);
    clusterCentroids(clustersFound,:) = bestCent(2,:);
    C(DsplitIDX) = newLabels;
end
% refinement step:
if doRefinement,
    refIDX = kmeans( D , k, 'start', clusterCentroids ) ;
    C = refIDX;
end
end
```

Listing B.43: evaluation/overallSimilarity.m

```
function os = overallSimilarity( D, IDX, CENT )
%OVERALLSIMILARITY Computes overall similarity for a clustering solution

% The overall similarity for the entire clustering solution is computed
% according to equation 3 in:
% Marcelo Nunes et al.: "Docs-Clustering: A System for Hierarchical
% Clustering and Document Labeling"; WTI 2008, pp.1-10
% (the actual similarity between a data point and a cluster centroid is not
% define there)
```

```
os  = 0.0;
nc  = size(CENT,1);
lex = unique(IDX);
if length(lex) ~= nc,
    error('Number␣of␣labels␣and␣centroids␣do␣not␣match');
end
for cx = 1:nc,
    % compute distance of each cluster element from centroid
    sumDist = 0;
    thisCluster = D(IDX==lex(cx),:);
    csz = size(thisCluster,1);
    for cix = 1: csz,
        if inf == norm((CENT(cx,:)- thisCluster(cix,:))),
            disp('inf');
        end
        sumDist = sumDist + norm((CENT(cx,:)- thisCluster(cix,:)));
    end
    os = os + (sumDist / csz);
end
os = os / nc;
end
```

Listing B.44: evaluation/assignClassesToClusters.m

```
function clusterClasses = assignClassesToClusters( clusters, classes)
%
% INPUT: clusters -- (N x 1) matrix, where a cluster (number) is assigned
%                    to each of the N elements
%                    -> hypothesis
%
%        classes  -- (N x 1) matrix, where a class (number) is assigned to
%                    each of the N elements
%                    -> reference, or, gold standard
%
%        *Q       -- the total number of clusters
%
%        *C       -- the total number of classes
%
% OUTPUT: clusterClasses -- (Q x C) matrix
%
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-02-15

if size(clusters,1) ~= size(classes,1),
    error(['Size␣mismatch␣between␣cluster␣assignments␣(',num2str(size(clusters,1)),↩
       ')␣and␣reference␣classes␣(',num2str(size(classes,1)),')']);
end
cLex = unique(classes);
qLex = unique(clusters);
```

```
Q = length(qLex);
C = length(cLex);
% create mapping from label to lexindex:
mQ = NaN(max(qLex),1);
mC = NaN(max(cLex),1);
for x=1:length(qLex),
    qLabel = qLex(x);
    mQ(qLabel) = x;
end
for x=1:length(cLex),
    cLabel = cLex(x);
    mC(cLabel) = x;
end
clusterClasses = zeros(Q, C);
% go through all frames
for f=1:size(clusters,1),
    thisq = clusters(f);
    thisc = classes(f);
    thisCol = mC(thisc);
    thisRow = mQ(thisq);
    % increment
    clusterClasses(thisRow, thisCol) = clusterClasses(thisRow, thisCol) + 1;
end
end
```

Listing B.45: evaluation/adjustedRandIndex.m

```
function ari = adjustedRandIndex( clusterClasses, varargin )
%ADJUSTEDRANDINDEX compute ARI according to Hubert & Arabie (1985)
%
% INPUT: clusterClasses -- (Q x C) matrix, where Q is the number of
%                          clusters and C is the number of classes in the
%                          gold standard. Each row corresponds to a cluster
%                          and each column corresponds to a gold-standard
%                          class. The values give the number of
%                          elements of the respective class in the cluster
% OPTIONAL:
%        c
%        n2
%
% Reference:
%   Hubert, L. & Arabie, P.
%   Comparing partitions Journal of Classification, 1985, 2, 193-218
% (definition of ari from page 198)
% Note: The contingency table in Hubert & Arabie's paper is (C x Q)!
nVarargs = length(varargin);
if nVarargs == 0
    c = sum(arrayfun(@m_chose, sum(clusterClasses)));
    n2 = nchoosek(sum(sum(clusterClasses)),2);
```

```
else
    % use pre-computed values:
    c  = varargin{1};
    n2 = varargin{2};
end
q   = sum(arrayfun(@m_chose, sum(clusterClasses,2)));
cqn = c*q/n2;
ari = ( sum(sum(arrayfun(@m_chose, clusterClasses))) -cqn  ) / (  0.5 * (  c+q ) - ↩
   cqn ) ;

end

% Helper function for the binomial coefficient:
function x = m_chose(n)
if n < 2,
    x = 0;
else
    x = nchoosek(n,2);
end
end
```

Listing B.46: evaluation/purity.m

```
function pur = purity( clusterClasses )
%PURITY Computes purity
%
% INPUT: clusterClasses -- (Q x C) matrix, where Q is the number of
%                          clusters and C is the number of classes in the
%                          gold standard. Each row corresponds to a cluster
%                          and each column corresponds to a gold-standard
%                          class, where the values give the number of
%                          elements of the respective class in the cluster
%
% OUTPUT: pur           -- purity value according to the definition from
%                          the source given below
%
% source:
% Introduction to Information Retrieval
% By Christopher D. Manning, Prabhakar Raghavan & Hinrich Schütze
% http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html
%
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-02-15

% rows: clusters
% column 1 : number of elements of majority class in that cluster
% column 2 : label of the majority class
maxClasses = zeros(size(clusterClasses,1), 2);
for qx = 1:size(clusterClasses,1),
```

```
    maxClasses(qx) = max(clusterClasses(qx,:)) ;
end
N   = sum(sum(clusterClasses));
pur = (1 / N) * sum(maxClasses(:,1)) ;
end
```

Listing B.47: evaluation/vmeasure.m

```
function v = vmeasure( A, beta )
%V-measure according to Rosenberg & Hirschberg, 2007.
%
% N      : total number of data points
% C      : set of classes { c_1, ..., c_n}
% K      : set of clusters {k_1, ..., k_m}
% A      : contingency table with
%          a_ij == number of data points which are
%                  members of class c_i and cluster k_j
% H(C|K) : conditional entropy
%
% INPUT:
%  A = (M x N)
%     rows: clusters,
%     columns: classes,
%     Note: this is transposed from the definition in the [R&H] paper
%           but consistent with Rosenberg's Java implementation [www]
%
%  beta (optional)
%     Default value: 1
%
% Sources:
% [R&H] Rosenberg, A. & Hirschberg, J.
%       V-Measure: A conditional entropy-based external cluster evaluation measure
%       In: Proceedings of the 2007 Joint Conference on Empirical Methods in ↩
   Natural Language Processing and Computational Natural Language Learning, ↩
   Association for Computational Linguistics, 2007, 410-420
%
% [www] http://eniac.cs.qc.cuny.edu/andrew/
%       http://eniac.cs.qc.cuny.edu/andrew/v-measure/ClusterEvaluator.tgz
%
N    = sum(sum(A));
numK = size(A,1);% number of clusters
numC = size(A,2);% number of classes
% [www] sets parameter beta to 1
if (nargin < 2),
    beta = 1;
end
classSum = sum(A);% column sums
clusterSum = sum(A,2);% row sums
% compute joint entropy:
```

```
%  H(C,K) = H(K,C)
%  H(C|K)        = H(K,C) - H(K)
%  H(C|K) + H(K) = H(K,C)
% and:
%  H(K|C)        = H(C,K) - H(C)
%  H(K|C) + H(C) = H(C,K)
% (1) homogeneity h
% compute H(C|K):
hck = 0;
for k = 1:numK,
    %clusterSum = sum(A(k,:),2) ;
    for c = 1:numC,
        ack = A(k,c);
        if ack ~= 0,
            % check if a_ck is 0
            % if ack==0, log(ack/clusterSum) == -Inf
            % Matlab treats 0*-Inf as NaN
            % and any number + NaN equals NaN,
            % so we get no result for H(C|K).
            % We thus assume here that 0 log(0) == 0.
            hck = hck - ((ack / N) * log( ack / clusterSum(k) ));
        end
    end
end
% compute H(C):
hc = 0;
for c = 1:numC,
    %ack = sum(A(:,c));
    p = classSum(c) / N;
    hc = hc - ( p * log(p) );
end
% (2) completeness c
% compute H(K|C)
hkc = 0;
for c = 1:numC,
    %classSum = sum(A(:,c)) ;
    for k = 1:numK,
        ack = A(k,c);
        if ack ~=0,
            hkc = hkc - (  (ack/N) * log(ack/classSum(c)) );
        end
    end
end
% compute H(K):
hk = 0;
for k = 1:numK,
    %ack = sum(A(k,:),2);
    p = clusterSum(k) / N;
    hk = hk - ( p * log(p) );
```

```
end
%joint == hkc + hc == hck + hk
joint = hkc + hc;
h = 1;
c = 1;
if joint ~= 0,
    h = 1 - (hck / hc);
    c = 1 - (hkc / hk);
end
if 0==h && 0==c,
    v=0;
else
    v =  (1+beta)*h*c  / ((beta*h)+c) ;
end
end
```

Listing B.48: evaluation/confusionEvaluation.m

```
function csvFile = confusionEvaluation( setup, allConfusions, allConfusionsIdx, ↩
   confusionLabel, speaker, dT, lex, clusterSz )
%CONFUSIONEVALUATION Evaluate confusion tables

numClass = length(lex);
fprintf('  Confusion and cluster size statistics ("%s"):\n', confusionLabel);
allConfusionsSum = sum(allConfusions, 3);
% Compute classification error:
% sum of off-diagonal entries / total sum of entries
cacc = classificationAccuracy(allConfusionsSum);
cerr = classificationError(allConfusionsSum);
fprintf('    Total classification accuracy: %5.3f (%8.6f)\n', cacc, cacc);
fprintf('    Total classification error:    %5.3f (%8.6f)\n', cerr, cerr);
% ---------------------------------------------------------------------
% Print statistics about average cluster size:
if ~isempty(allConfusionsIdx),
    % -- allConfusions contains the "sorted" confusion matrices --
    % Multiple rows may correspond to the same class -- we need to
    % determine the majority class for each row:
    clusterSizes = cell(numClass,1);
    for it=1:setup.RUNS,
        thisSizes = sum(allConfusions(:,:,it),2);
        thisClasses = allConfusionsIdx(:,:,it);
        for x=1:numClass,
            clz  = thisClasses(x);
            clusterSizes{clz} = [clusterSizes{clz} thisSizes(x)];
        end
    end
    stdClusterSize  = zeros(numClass,1);
    avgClusterSizes = zeros(numClass,1);
    for x=1:numClass,
```

```
        stdClusterSize(x)  = std(clusterSizes{x});
        avgClusterSizes(x) = sum(clusterSizes{x}) / length(clusterSizes{x});
    end
    avgClusterSize = sum(avgClusterSizes) / numClass;
else
    % -- allConfusions contains the "matched" confusion matrices --
    % Create temp table to compute std for each cluster:
    % rows: iterations; columns: clusters
    clusterSizes = zeros(setup.RUNS,numClass);
    for it =1:setup.RUNS,
        clusterSizes(it,:) = sum(allConfusions(:,:,it),2);
    end
    stdClusterSize = std(clusterSizes);
    summedClusterSizes = sum(allConfusionsSum,2);
    avgClusterSizes    = summedClusterSizes / setup.RUNS;
    avgClusterSize     = sum(summedClusterSizes) / (numClass * setup.RUNS);
end
fprintf('␣␣␣␣Cluster␣size␣statistics:\n');
clusterSizeErr = zeros(numClass,1);
for c=1:numClass,
    avgSz = avgClusterSizes(c); %summedClusterSizes(c) / setup.RUNS;
    clusterSizeErr(c) = abs(avgSz - clusterSz);
    fprintf('␣␣␣␣␣%3s-cluster:␣avg=␣%7.3f;␣␣std=␣%7.3f\n', lex{c}, avgSz, ↩
       stdClusterSize(c));
end
avgStd = sum(stdClusterSize)/numClass;% average std. dev.
fprintf('␣␣␣␣Overall␣avg.␣cluster␣size:␣%7.3f\n', avgClusterSize );
fprintf('␣␣␣␣Overall␣std.␣cluster␣size:␣%7.3f\n', avgStd  );
avgSzErr =  sum(clusterSizeErr) / (clusterSz * numClass);
fprintf('␣␣␣␣Total␣cluster␣size␣error:␣␣%5.3f␣␣(%8.6f)\n',  avgSzErr, avgSzErr   );
% ---------------------------------------------------------------------
% Store results to files for later processing
rowlex = cell(size(lex));
for x=1:length(lex),
    rowlex{x} = [lex{x} '-cluster'];
end
outFile = [setup.baseDir speaker '_' setup.dataLabels{dT} '_confusions', ↩
   confusionLabel, '.tab'];
printMatrixToTab( outFile, allConfusionsSum, lex, rowlex, false, false, false );
fprintf('␣␣␣␣Stored␣confusions␣to:␣%s\n', outFile);
csvFile = [setup.baseDir speaker '_' setup.dataLabels{dT} '_confusions', ↩
   confusionLabel, '.csv'] ;
printMatrixToTextfile(csvFile , allConfusionsSum, lex, rowlex, '%d', ',', '\n', ↩
   false, false, false, false ) ;
% ---------------------------------------------------------------------
% Compute broad confusions:
if ~isempty(setup.broadClasses),
    fprintf('␣␣␣␣Broad␣classification:\n')
    numBC = length(setup.broadClasses);
```

```
    broadConf = zeros(numBC,numBC);
    for c=1:numBC,
        for cx=1:numBC,
            broadConf(c,cx) = sum(sum(allConfusionsSum(setup.broadClasses{c}, setup.broadClasses{cx})));
        end
    end
    cacc = classificationAccuracy(broadConf);
    cerr = classificationError(broadConf);
    fprintf('     Classification accuracy on broad classes: %5.3f (%8.6f)\n', cacc, cacc);
    fprintf('     Classification error on broad classes:    %5.3f (%8.6f)\n', cerr, cerr);
    outFile = [setup.baseDir speaker '_' setup.dataLabels{dT} '_broadConfusions', confusionLabel, '.tab'];
    printMatrixToTab( outFile, broadConf, setup.broadClassLabels, setup.broadClassLabels, false, false, false );
    fprintf('    Stored confusions to: %s\n\n', outFile);
    csvFileBC = [setup.baseDir speaker '_' setup.dataLabels{dT} '_broadConfusions', confusionLabel, '.csv'] ;
    printMatrixToTextfile(csvFileBC , broadConf, setup.broadClassLabels, setup.broadClassLabels, '%d', ',', '\n', false, false, false, false ) ;
end
end
```

Listing B.49: evaluation/r.m

```
function [result status] = r(varargin)
%R Execute a R command
%
% This function is based on Matlab's original "Perl" function
%
% [result status] = r(file, arg1 ... argN)
%  INPUT: file                    - path and filename of an R script file
%         arg1...argN (optional) - arguments to be passed to the R script
%                                  the R command line option --args is
%                                  appended automatically as necessary
%
% See also:
% http://cran.r-project.org/doc/manuals/R-intro.html#Invoking-R-from-the-command-line

%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-07-07
RCALL = 'R --no-restore --slave';
RFILE = '';
cmdString = '';
% Add input to arguments to operating system command to be executed.
% (If an argument refers to a file on the MATLAB path, use full file path.)
```

```
for i = 1:nargin
    thisArg = varargin{i};
    if isempty(thisArg) || ~ischar(thisArg)
        error('All input arguments must be valid strings.');
    end
    if i==1
        RFILE = thisArg;
        if exist(RFILE, 'file')==2
            % This is a valid file on the MATLAB path
            if isempty(dir(RFILE))
                % Not complete file specification
                % - file is not in current directory
                % - OR filename specified without extension
                % ==> get full file path
                RFILE = which(RFILE);
            end
        else
            % First input argument is R script file - it must be a valid file
            error( 'Unable to find R file: %s', RFILE);
        end
    else
        % Wrap thisArg in double quotes if it contains spaces
        if any(thisArg == ' ')
            thisArg = ['"', thisArg, '"'];
        end

        % Add argument to command string
        cmdString = [cmdString, ' ', thisArg];
    end
end
% Execute R script
if ~isempty(cmdString),
    cmdString = [' --args ', cmdString];
end
[status ignore] = unix('which R'); %#ok
if (status == 0)
    cmdString = [RCALL, cmdString, ' < ', RFILE ];
    [status, result] = unix(cmdString);
else
    error('Unable to find R executable.');
end
% Check for errors in shell command
if nargout < 2 && status~=0
    error('System error: %sCommand executed: %s', result, cmdString);
end
```

Listing B.50: printConfusionMatrix.R

```
# Daniel Duran, Uni Stuttgart, IMS / SFB 732, A2
```

```
#
# Evaluation and visualization of results for the "EMA clustering" experiments
# to be submitted to INTERSPEECH 2011
#
# File created: 2011-03-15
#       edited: 2011-03-23 using "fileNames" vector
#               2011-03-28
#               2011-07-07 moved to ~/myCode/diss/emaClustering/R/↩
   printConfusionMatrix.R
#                          fixed font embedding in PDF
#                          added command line arguments for file names
#
# call e.g. > source("printConfusionMatrix.R")
#

# get command line arguments
args <- commandArgs(trailingOnly=TRUE);

if(length(args) != 4){
 stop("Missing␣arguments.␣Required:␣2␣input␣(EMA;␣ENV␣)␣and␣2␣output␣files!");
}

emaFile    <- args[1];
envFile    <- args[2];

outFileRel <- args[3];
outFileAbs <- args[4];

ema    <- read.csv(emaFile);
env    <- read.csv(envFile);

maxVal = 0;

    pdf(file=outFileRel, colormodel="cmyk") ;
    par(mar=c(0.1,4.1,0.4,0.1));

    layout(matrix(1:nrow(ema),nrow(ema),1,byrow=TRUE),
   heights=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.7));

    N = nrow(ema);

    for ( r in 1:N )
    {
   ddist = matrix(rep(0,times=N*2),2);

   cDist = ema[r,];
   m = max(ema[r,]);
   if (m > maxVal){
     maxVal = m;
```

```
50    }
51    cDist = cDist * 100 / sum(cDist);
52    ddist[1,] = unlist(cDist, use.names=FALSE);
53
54    cDist = env[r,];
55    m = max(env[r,]);
56    if (m > maxVal){
57      maxVal = m;
58    }
59    cDist = cDist * 100 / sum(cDist);
60    ddist[2,] = unlist(cDist, use.names=FALSE);
61
62
63    if(r==N){
64       par( mar=c(2.1,4.1,0.4,0.1) );
65       barplot(ddist, beside=TRUE,
66          space=c(0,0.5),
67          ylim=c(0,100),
68          names.arg=colnames(ema),
69          cex.names=1.5
70       );
71
72    } else {
73       barplot(ddist, beside=TRUE,
74          space=c(0,0.5),
75          ylim=c(0,100)
76       );
77    }
78    mtext(rownames(ema)[r], side=2, line=3) ;
79
80     }
81     dev.off() ;
82     call = embedFonts(outFileRel,format="pdfwrite", options="-dPDFSETTINGS=/↩
          prepress␣-dEPSCrop") ;
83
84 # Print matrix with absolute values
85
86     pdf(file=outFileAbs, colormodel="cmyk") ;
87
88
89     par( #las=1,
90    mar=c(0.1,4.1,0.4,0.1)  # mar=c(5.1, 6.1, 4.1, 2.1),
91             # 'c(bottom, left, top, right)'
92             # ... The default is 'c(5, 4, 4, 2) + 0.1'
93    #mfrow = c(nrow(clusterClasses),1)
94     );
95
96     layout(matrix(1:nrow(ema),nrow(ema),1,byrow=TRUE),
97    heights=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.7));
```

```

 N = nrow(ema);

 for ( r in 1:N )
 {
ddist = matrix(rep(0,times=N*2),2);

cDist = ema[r,];
#cDist = cDist * 100 / sum(cDist);
ddist[1,] = unlist(cDist, use.names=FALSE);

cDist = env[r,];
#cDist = cDist * 100 / sum(cDist);
ddist[2,] = unlist(cDist, use.names=FALSE);


if(r==N){
   par( mar=c(2.1,4.1,0.4,0.1) );
   barplot(ddist, beside=TRUE,
      space=c(0,0.5),
      ylim=c(0,maxVal),
      names.arg=colnames(ema),
      cex.names=1.5
   );

} else {
   barplot(ddist, beside=TRUE,
      space=c(0,0.5),
      ylim=c(0,maxVal)
   );
}
mtext(rownames(ema)[r], side=2, line=3) ;
 }

 dev.off() ;
 call = embedFonts(outFileAbs,format="pdfwrite", options="-dPDFSETTINGS=/prepress -dEPSCrop") ;
```

B.2.2. Experiment 1: baseline

The source code for this experiment is identical to the one in section B.2.1, except for the two files listed here: the first one setting the parameters and the second one defining the a random clustering function.

Listing B.51: clusteringExperiment1baseline.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%%
```

```
%%% Baseline method: random cluster assignments

%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('evaluation');
addpath('../io');

THISMFILENAME = 'clusteringExperiment1baseline';
% ==============================================================================
% Initialize experiment setup
% ------------------------------------------------------------------------------
setup = struct([]);
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/clustering/';
setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};
% the data types to use in the experiment:
% first column: articulatory data EMA
% second column: audio data ACC
% -> (1) EMA (2) ACC (3) EMA+ACC
setup.dataTypes = [1 0 ; 0 1 ; 1 1];
setup.dataLabels = {'EMA', 'ENV', 'EMAENV'};
% % Audio representation
setup.useENVB   = true; % use amplitude envelope bands
setup.useMFCC   = false;% use MFCC
setup.useWAVE   = false;% (using the raw wave signal is not supported)
% Delta1 and Delta2 are the signals' first and second derivatives:
setup.useDelta1 = true;
setup.useDelta2 = true;
% ------------------------------------------------------------------------------
% Function handle for the function which reads the data from the
% pre-processed corpus files and create the data structures used in the
% clustering experiment:
setup.dataImportFH = @getClusteringDataFromPolishEMA;
% ------------------------------------------------------------------------------
% Function handle for the function which takes a sample set from the
% corpus:
setup.samplingFH = @getSampleSet;
% ------------------------------------------------------------------------------
% Set how many samples per class should be exported as a ratio of the smallest
% phone class for a speaker, e.g. 0.75 means that the exported classes will all
% be 75% of the size of the smallest phone class
setup.minClusterRatio = 0.75;
% Set the number of repeated runs of the clustering procedure on a sample set:
setup.RUNS = 1000;
% Function handle for the actual clustering function
% here: a random clustering function
setup.clusterFH = @randomClustering ;
```

```
% Define sets of classes to evaluate confusions between "broad phonetic
% classes":
setup.broadClasses = {[1 2], [3 4], [5 6 7], 8 };
setup.broadClassLabels = {'K', 'R', 'Y', 'A'};
% ---------------------------------------------------------------------------
% Evaluation
% set path and file name for the printing R script:
% (MATLAB cannot embedd fonts into PDF files)
setup.RSCRIPT  = '../../r/printConfusionMatrix.R';
setup.RSCRIPT2 = '../../r/printConfusionMatrix2.R';
% ---------------------------------------------------------------------------
% print debugging info to console (and log file)
setup.VERBOSE=true;
% ---------------------------------------------------------------------------
% set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, setup.timestamp, '/' ] ;
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
% ===========================================================================
%% run %%%
% Save terminal output to log file and display current settings
diary(diaryFilename);
disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
disp(['pwd␣=␣', pwd]);
disp( '␣␣Setup:␣');
disp(setup);
% this runs the actual experiment:
experiment1(setup);
%% finished
disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣', ↩
    setup.timestamp, '.␣Bye!']);
diary off;
```

Listing B.52: functions/randomClustering.m

```
function [ IDX ] = randomClustering( D, k, varargin )
%RANDOMCLUSTERING assigns randomly cluster labels 1:k to the rows in D
n   = ceil( size(D,1)/k);
IDX = shuffle( repmat([1:k]',n,1) );
IDX = IDX(1:size(D,1));
end
```

B.2.3. Experiment 1: refinement with contiguous sequences

Listing B.53: clusteringExperiment1c.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% based on: "exportForClustering.m"  [2010-12-07]
%%%           "clusteringEvaluation.m" [2010-12-07]
%%%
%%%           "code/csm_EMMA/exportForClusteringEq.m"          [2011-03-22]
%%%           "code/csm_EMMA/clusteringEvaluationCombinedEq.m" [2011-03-23]
%%%
%%% Reference: Duran et al. Interspeech, 2011
%%%

%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('evaluation');
addpath('../io');
THISMFILENAME = 'clusteringExperiment1c';
% ==============================================================================
% Initialize experiment setup
% ------------------------------------------------------------------------------
setup = struct([]);
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/clustering/';
setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};
% the data types to use in the experiment:
% first column: articulatory data EMA
% second column: audio data ACC
% -> (1) EMA (2) ACC (3) EMA+ACC
setup.dataTypes = [1 0 ; 0 1 ; 1 1];
setup.dataLabels = {'EMA', 'ENV', 'EMAENV'};
% % Audio representation
setup.useENVB   = true; % use amplitude envelope bands
setup.useMFCC   = false;% use MFCC
setup.useWAVE   = false;% (using the raw wave signal is not supported)
% Delta1 and Delta2 are the signals' first and second derivatives:
setup.useDelta1 = true;
setup.useDelta2 = true;
% ------------------------------------------------------------------------------
% Function handle for the function which reads the data from the
% pre-processed corpus files and create the data structures used in the
% clustering experiment:
setup.dataImportFH = @getClusteringDataFromPolishEMA;
% ------------------------------------------------------------------------------
% Function handle for the function which takes a sample set from the
% corpus:
setup.samplingFH = @getSampleSetCont;
setup.MAXSEQLENGTH = 25;
% ------------------------------------------------------------------------------
```

```
% Set how many samples per class should be exported as a ratio of the smallest
% phone class for a speaker, e.g. 0.75 means that the exported classes will all
% be 75% of the size of the smallest phone class
setup.minClusterRatio = 0.75;
% Set the number of repeated runs of the clustering procedure on a sample set:
setup.RUNS = 1000;
setup.clusterFH = @bisectingKmeans ;
% Number of iterations of the bisecting k-means clustering:
setup.BKMiter = 10;
% Use cluster refinement in bisecting k-means:
setup.BKMrefine = true;
% Define sets of classes to evaluate confusions between "broad phonetic
% classes":
setup.broadClasses = {[1 2], [3 4], [5 6 7], 8 };
setup.broadClassLabels = {'K', 'R', 'Y', 'A'};
% ---------------------------------------------------------------------------
% Evaluation
% set path and file name for the printing R script:
% (MATLAB cannot embedd fonts into PDF files)
setup.RSCRIPT  = '../../r/printConfusionMatrix.R';
setup.RSCRIPT2 = '../../r/printConfusionMatrix2.R';
% ---------------------------------------------------------------------------
% print debugging info to console (and log file)
setup.VERBOSE=true;
% ---------------------------------------------------------------------------
% set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, setup.timestamp, '/' ] ;
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
% ===========================================================================
%% run %%%
% Save terminal output to log file
diary(diaryFilename);
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp(['pwd␣=␣', pwd]);
    disp( '␣␣Setup:␣');
    disp(setup);
end
% this runs the actual experiment:
experiment1(setup);
%% finished
disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣', ↩
   setup.timestamp, '.␣Bye!']);
diary off;
```

Listing B.54: functions/getSampleSetCont.m

```
function [ Deq, IDXeq ] = getSampleSetCont( setup, D, IDX, numClass, clusterSz )
% Create sample with equally represented classes and contiguous
% sequences of frames from the same class
rows  = length(IDX);
indicesEq = zeros(rows, numClass);
done = zeros(rows,1);
for c = 1:numClass,
    % assuming that the class indices form an incremental list
    % starting at 1
    % get all frame indices from the current phone class:
    currentClass = find( IDX == c) ;
    % shuffle the indices:
    sx = shuffle(currentClass);
    found = 0;
    ptrSx =1;
    while found < clusterSz,
        % take the next random frame index
        ix = sx(ptrSx);
        ptrSx = ptrSx + 1;
        if done(ix),
            % This frame has already been taken for the current
            % sample set
            continue,
        end
        indicesEq(ix,c) = 1;
        done(ix) = 1;
        found = found + 1;
        if found < clusterSz,
            for cx=ix+1:(ix+setup.MAXSEQLENGTH),
                if cx > rows || IDX(cx) ~= c,
                    break;
                end
                if done(cx),
                    continue;
                end
                indicesEq(cx,c) = 1;
                done(cx) = 1;
                found = found + 1;
                if found >= clusterSz,
                    break;
                end
            end
        end
    end
end
indices = find( sum(indicesEq,2) );
Deq   = D(indices,:);
```

```
IDXeq = IDX(indices);
end
```

B.2.4. Experiment 2

Listing B.55: clusteringExperiment2.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% based on:
%%%
%%% Reference: Duran et al. LabPhon, 2012

%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('evaluation');
addpath('../io');
THISMFILENAME = 'clusteringExperiment2';

%     k:  1327  ( 1)
%     p:  1477  ( 2)
%     r:   593  ( 3)
%     l:   960  ( 4)
%     u:   154  ( 5)
%     y:   340  ( 6)
%     i:   622  ( 7)
%     a:  1902  ( 8)
TARGETS = { [7]; [6] };
CLASSES = { {[7 1]; [7 1 3]}; {[6 2]; [6 2 3]} };% [t c1 c2] for VCC experiments
STARTTIME = datestr(now,'yyyy-mm-dd+HH-MM');
for ex=1:length(TARGETS),
    % ==========================================================================
    % Initialize experiment setup
    % --------------------------------------------------------------------------
    setup = struct([]);
    setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
    setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/clustering/';
    setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
    setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};
    % The data types to use in the experiment: specified by a matrix, with
    %  first column: articulatory data EMA;
    %  second column: audio data ACC
    %  -> (1) EMA (2) ACC (3) EMA+ACC
    setup.dataTypes = [1 0 ; 0 1 ; 1 1];
    %  labels used for output files:
    setup.dataLabels = {'EMA', 'ACC', 'EMA+ACC'};
    % Selecting a specific audio representation:
```

```
41      setup.useENVB   = true; % use amplitude envelope bands
42      setup.useMFCC   = false;% use MFCC
43      setup.useWAVE   = false;% (using the raw wave signal is not supported)
44      % Delta1 and Delta2 are the signals' first and second derivatives:
45      setup.useDelta1 = true;
46      setup.useDelta2 = true;
47      % Define the set of target segments which should be exrtaced from the
48      % corpus for clustering:
49      setup.targets = TARGETS{ex};% T target segments
50      setup.targetClasses = CLASSES{ex};
51      % Handle for the function which determines the reference class for a
52      % given target vowel:
53      setup.classGetterFH = @getTargetClass;
54      setup.broadClasses = {};
55      % ---------------------------------------------------------------------
56      % Function handle for the function which reads the data from the
57      % pre-processed corpus files and create the data structures used in the
58      % clustering experiment:
59      setup.dataImportFH = @getVowelClusteringDataFromPolishEMA;
60      % ---------------------------------------------------------------------
61      % Function handle for the function which takes a sample set from the
62      % corpus:
63      setup.samplingFH = @getSampleSet;
64      % ---------------------------------------------------------------------
65      % Set how many samples per class should be exported as a ratio of the
66      % smallest phone class for a speaker, e.g. 0.75 means that the exported
67      % classes will all be 75% of the size of the smallest phone class
68      setup.minClusterRatio = 0.75;
69      % Set the number of repeated runs of the clustering procedure on a sample set:
70      setup.RUNS = 1000;
71      % ---------------------------------------------------------------------
72      % Set handle to a clustering function:
73      setup.clusterFH = @bisectingKmeans;
74      % The following parameters are relevant only for bisecting k-means
75      % clustering:
76      % Number of iterations:
77      setup.BKMiter = 10;
78      % Use cluster refinement: {true,false}
79      setup.BKMrefine = true;
80      % ---------------------------------------------------------------------
81      % Evaluation
82      % set path and file name for the printing R script:
83      % (MATLAB cannot embedd fonts into PDF files)
84      setup.RSCRIPT  = '../../r/printConfusionMatrix.R';
85      setup.RSCRIPT2 = '../../r/printConfusionMatrix2.R';
86      % ---------------------------------------------------------------------
87      % print debugging info to console (and log file)
88      setup.VERBOSE=true;
89      % ---------------------------------------------------------------------
```

```
    % Set and create the base directory for this experiment's output:
    setup.baseDir = [setup.OUTDIR, setup.timestamp, '/' ] ;
    % Save terminal output to a log file
    diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
    % Create basic output directory for all parameter combindations:
    mkdir(setup.baseDir);
    % ========================================================================
    %% run %%%
    % Start logging and print experiment setup:
    diary(diaryFilename);
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp(['pwd␣=␣', pwd]);% current working directory
    disp( '␣␣Setup:␣');
    disp(setup);
    % This runs the actual experiment:
    experiment1(setup);
    %% finished
    disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣'↩
        , setup.timestamp, '.␣Bye!']);
    diary off;% close log file
end % for ex
fprintf('Experiment␣2:␣done.␣Start␣time␣was:␣%s\n', STARTTIME );
```

Listing B.56: clusteringExperiment2C.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% based on:
%%%
%%% Reference: Duran et al. LabPhon, 2012

%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('evaluation');
addpath('../io');
THISMFILENAME = 'clusteringExperiment2C';
%     k:  1327  ( 1)
%     p:  1477  ( 2)
%     r:   593  ( 3)
%     l:   960  ( 4)
%     u:   154  ( 5)
%     y:   340  ( 6)
%     i:   622  ( 7)
%     a:  1902  ( 8)
PHONGROUPS = {[1 3 4]; [1]; [3]; [2]; [1 2]; [3 4]};% krl, k, r, p, pk, rl
TARGETS = { [7]; [6] };
CLASSES = { {[7 1]; [7 2 3]}; ... % tik,gil,tir vs. wikr
    {[6 4]; [6 5 6]} }; % typ vs. cypr, cykl
```

```
24 % [t c1 c2] for VCC experiments
25 STARTTIME = datestr(now,'yyyy-mm-dd+HH-MM');
26 for ex=1:length(TARGETS),
27     % ==========================================================================
28     % Initialize experiment setup
29     % --------------------------------------------------------------------------
30     setup = struct([]);
31     setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
32     setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/clustering/';
33     setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
34     setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};
35     % The data types to use in the experiment: specified by a matrix, with
36     %  first column: articulatory data EMA;
37     %  second column: audio data ACC
38     %  -> (1) EMA (2) ACC (3) EMA+ACC
39     setup.dataTypes = [1 0 ; 0 1 ; 1 1];
40     %  labels used for output files:
41     setup.dataLabels = {'EMA', 'ACC', 'EMA+ACC'};
42     % Selecting a specific audio representation:
43     setup.useENVB   = true; % use amplitude envelope bands
44     setup.useMFCC   = false;% use MFCC
45     setup.useWAVE   = false;% (using the raw wave signal is not supported)
46     % Delta1 and Delta2 are the signals' first and second derivatives:
47     setup.useDelta1 = true;
48     setup.useDelta2 = true;
49     % Define the set of target segments which should be exrtaced from the
50     % corpus for clustering:
51     setup.targets = TARGETS{ex};% T target segments
52     setup.targetClasses = CLASSES{ex};
53     setup.targetGroups = PHONGROUPS;
54     % Handle for the function which determines the reference class for a
55     % given target vowel:
56     setup.classGetterFH = @getTargetClassC;
57     setup.broadClasses = {};
58     % --------------------------------------------------------------------------
59     % Function handle for the function which reads the data from the
60     % pre-processed corpus files and create the data structures used in the
61     % clustering experiment:
62     setup.dataImportFH = @getVowelClusteringDataFromPolishEMA;
63     % --------------------------------------------------------------------------
64     % Function handle for the function which takes a sample set from the
65     % corpus:
66     setup.samplingFH = @getSampleSet;
67     % --------------------------------------------------------------------------
68     % Set how many samples per class should be exported as a ratio of the
69     % smallest phone class for a speaker, e.g. 0.75 means that the exported
70     % classes will all be 75% of the size of the smallest phone class
71     setup.minClusterRatio = 0.75;
72     % Set the number of repeated runs of the clustering procedure on a sample set:
```

```
    setup.RUNS = 1000;
    % ---------------------------------------------------------------------
    % Set handle to a clustering function:
    setup.clusterFH = @bisectingKmeans;
    % The following parameters are relevant only for bisecting k-means
    % clustering:
    % Number of iterations:
    setup.BKMiter = 10;
    % Use cluster refinement: {true,false}
    setup.BKMrefine = true;
    % ---------------------------------------------------------------------
    % Evaluation
    % set path and file name for the printing R script:
    % (MATLAB cannot embedd fonts into PDF files)
    setup.RSCRIPT  = '../../r/printConfusionMatrix.R';
    setup.RSCRIPT2 = '../../r/printConfusionMatrix2.R';
    % ---------------------------------------------------------------------
    % print debugging info to console (and log file)
    setup.VERBOSE=true;
    % ---------------------------------------------------------------------
    % Set and create the base directory for this experiment's output:
    setup.baseDir = [setup.OUTDIR, setup.timestamp, '/' ] ;
    % Save terminal output to a log file
    diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
    % Create basic output directory for all parameter combindations:
    mkdir(setup.baseDir);
    % =====================================================================
    %% run %%%
    % Start logging and print experiment setup:
    diary(diaryFilename);
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp(['pwd␣=␣', pwd]);% current working directory
    disp( '␣␣Setup:␣');
    disp(setup);
    % This runs the actual experiment:
    experiment1(setup);

    %% finished
    disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣'↩
        , setup.timestamp, '.␣Bye!']);
    diary off;% close log file
end % for ex
fprintf('Experiment␣2C:␣done.␣Start␣time␣was:␣%s\n', STARTTIME );
```

Listing B.57: functions/getVowelClusteringDataFromPolishEMA.m

```
function [ D, IDX, lex ] = getVowelClusteringDataFromPolishEMA( setup, speakerNr, ↩
    useACC, useART )
```

```
% Get data from the Polish EMA corpus for clustering experiment 2
%
% OUTPUT:
%  D   = (N x d)  data set of N d-dimensional data points
%
%  IDX = (N x 1)  class index labels for each data point (reference
%                 classification used for external evaluation)
%
%  lex = {C x 1}  cell array with string labels for each of the C classes
%
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart

% ----------------------------------------------------------------------
% load index and label data for this speaker:
inDir = [setup.matlabDir setup.speakerDirs{speakerNr}];
load([inDir 'index.mat']);

% check whether the variables are loaded successfully
if ~exist('baseNames','var') || ...
        ~exist('totalFramesEma','var') || ~exist('totalFramesWav','var')  || ...
        ~exist('totalFramesEnv','var') || ~exist('totalFramesMFCC','var') || ...
        ~exist('segmentsPHON','var')   || ~exist('segmentsWORD','var') || ...
        ~exist('corpusFs','var'),
    % add more checks?
    error('index.mat␣was␣not␣opened␣successfully!');
end
% ----------------------------------------------------------------------
%%
% Determine the number of columns for the output data structurs (each
% column corresponds to one "dimension" of the data):
offset=0;
accCols = [];
emaCols = [];
envCols = [];
mfcCols = [];
if useACC,
    if setup.useWAVE,
        disp('WARNING:␣adding␣WAVE␣not␣implemented!');
    end
    if setup.useMFCC,
        plus = 13;
        if setup.useDelta1 && setup.useDelta2,
            plus=39;
        else
            if setup.useDelta1 || setup.useDelta2,
                plus=26;
            end
        end
        mfcCols = (offset+1) : (offset+plus);% the MFCC columns in "corpus"
```

```
        offset = offset + plus;
    end
    if setup.useENVB,
        % use amplitude envelop representation (default for audio)
        plus = 8;
        if setup.useDelta1 && setup.useDelta2,
            plus=24;
        else
            if setup.useDelta1 || setup.useDelta2,
                plus=16;
            end
        end
        envCols = (offset+1) : (offset+plus);% the envelop columns in "corpus"
        offset = offset + plus;
    end
    accCols = [mfcCols envCols];
end
if useART,
    plus = 8;
    if setup.useDelta1 && setup.useDelta2,
        plus=24;
    else
        if setup.useDelta1 || setup.useDelta2,
            plus=16;
        end
    end
    emaCols = (offset+1) : (offset+plus);
    offset = offset + plus;
end

%%
% ---------------------------------------------------------------------------
% Initialize output data
rows   = max([totalFramesEma, totalFramesEnv, totalFramesMFCC]) ;
D   = zeros(rows, offset);
IDX = zeros(rows,1);
lex = cell(size(setup.targetClasses));
ptr    = 1 ;
ptrEnd = 0;
% loop through utterances (files)
for fdx=1:length(baseNames), %5, %
    bfile = baseNames{fdx};
    skip  = false;
    % Check if the required data is available:
    if useART,
        if sum(checkFiles(fdx,1:4)) ~= 4,
            disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣EMA␣data']);
            skip = true ;
        end
```

```
    end
    if useACC || setup.useWAVE || setup.useMFCC,
        if checkFiles(fdx,5) ~= 1,
            disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣Audio␣data']);
            skip = true ;
        end
    end
    if skip,
        continue;
    end
    % the segments of this utterance:
    seg = segmentsPHON( segmentsPHON(:,4)==fdx , :);

    if length(find( seg(:,3) )) ~= size(seg,1) ,
        disp(['Warning:␣unlabeled␣segment␣found␣in␣' baseNames{fdx}, '␣(fdx=', ↩
           num2str(fdx), ')␣-->␣skipping!']);
        continue;
    end
    % Check if this utterance contains one of the target segments
    [uttTarget, uttPos, uttClz, clzLab, lex] = setup.classGetterFH(setup, seg, ↩
       lexPHON, lex);
    if 0>=uttTarget,
        continue;
    end
    dataART = [];
    dataACC = [];
    idxACC  = [];
    idxART  = [];
    % == EMA ======================================================== EMA ==
    if useART,
        % Add articulatory data from corpus
        % load EMA data for the current utterance:
        load([inDir bfile '_ema.mat']);

        from = seg(uttPos,1);
        to   = seg(uttPos,2);

        [dataART from to] = m_getChunk(from, to, true, setup.useDelta1, setup.↩
           useDelta2, ...
            signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
            signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy );

        idxART = repmat(uttClz, size(dataART,1), 1);

    end % if useART
    % == ENV ======================================================== ENV ==
    if useACC,
        % load ENV data for the current utterance:
        load([inDir bfile '_audio.mat']);
```

```
146         from = seg(uttPos,1);
147         to   = seg(uttPos,2);
148         if setup.useMFCC,
149             [lcM from to] = m_getMFCCChunk(from, to, signalMFCC, ...
150                 setup.useDelta1, setup.useDelta2, true );
151         else
152             lcM=[];
153         end
154         if setup.useENVB,
155             [lcE from to] = m_getEnvChunk(from, to, signalENVB, ...
156                 setup.useDelta1, setup.useDelta2, true );
157         else
158             lcE=[];
159         end
160         dataACC = [lcM lcE];
161         idxACC = repmat(uttClz, size(dataACC,1), 1);
162     end % if useACC
163     % check if all required data was found and if it is consistent
164     % if yes: add to D
165     % if no:  dismiss and skip this utterance
166     if useACC && useART,
167         if (size(dataACC,1) ~= size(dataART,1)),
168             disp('Warning: data lengths do not match!');
169             ptrEnd = ptr + min([size(dataACC,1), size(dataART,1)]) - 1;
170         else
171             ptrEnd = ptr + size(dataACC,1) - 1;
172         end
173         if ~isempty(find(idxART~=idxACC, 1)),
174             disp('Warning: labels mismatch. Skipping!');
175             skip=true;
176         end
177     else
178         % dataACC or dataART is empty
179         ptrEnd = ptr + max([size(dataACC,1), size(dataART,1)]) - 1;
180     end
181     if skip,
182         continue;
183     end
184     if useACC,
185         D(ptr:ptrEnd,accCols) = dataACC;
186         IDX(ptr:ptrEnd) = idxACC;
187     end
188     if useART,
189         D(ptr:ptrEnd,emaCols) = dataART;
190         IDX(ptr:ptrEnd) = idxART;
191     end
192     ptr = ptrEnd + 1;
193 end % for fdx
194 if ptrEnd < rows,
```

```
    % truncate output data to actual length
    D   = D(1:ptrEnd,:);
    IDX = IDX(1:ptrEnd);
end
disp('Class␣label␣lexicon:');
for x=1:length(lex),
    fprintf('␣(%d)␣%s\n', x, lex{x});
end
fprintf('[%s␣%s]␣Finished␣data␣import.␣Total␣number␣of␣samples:␣%d\n', datestr(now↩
   ,31), mfilename, ptrEnd );
end

% ==========================================================================
%% Function: m_getChunk
% get chunk from the signal in the range [from:to]
function [out from to] = m_getChunk(from, to, VERBOSE, useDelta1, useDelta2, ...
    signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
    signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE', ['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalLLIPx),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalLLIPx)), '!' ]);
    end
    to = length(signalLLIPx);
end
if useDelta1 && useDelta2,
    dims = 24;
else
    if useDelta1 || useDelta2,
        dims = 16;
    else
        dims = 8;
    end
end
out = zeros(to - from +1, dims);
out(:,1) = signalLLIPx(from:to,1);
out(:,2) = signalLLIPy(from:to,1);
out(:,3) = signalTBO1x(from:to,1);
out(:,4) = signalTBO1y(from:to,1);
out(:,5) = signalTBO2x(from:to,1);
out(:,6) = signalTBO2y(from:to,1);
```

```
243 out(:,7) = signalTTIPx(from:to,1);
244 out(:,8) = signalTTIPy(from:to,1);
245 if useDelta1,
246     out(:,9)  = signalLLIPx(from:to,2);
247     out(:,10) = signalLLIPy(from:to,2);
248     out(:,11) = signalTBO1x(from:to,2);
249     out(:,12) = signalTBO1y(from:to,2);
250     out(:,13) = signalTBO2x(from:to,2);
251     out(:,14) = signalTBO2y(from:to,2);
252     out(:,15) = signalTTIPx(from:to,2);
253     out(:,16) = signalTTIPy(from:to,2);
254 end
255 if useDelta2,
256     out(:,17) = signalLLIPx(from:to,3);
257     out(:,18) = signalLLIPy(from:to,3);
258     out(:,19) = signalTBO1x(from:to,3);
259     out(:,20) = signalTBO1y(from:to,3);
260     out(:,21) = signalTBO2x(from:to,3);
261     out(:,22) = signalTBO2y(from:to,3);
262     out(:,23) = signalTTIPx(from:to,3);
263     out(:,24) = signalTTIPy(from:to,3);
264 end
265 end
266
267 % ========================================================================
268 %%
269 function [out from to] = m_getEnvChunk(from, to, signalENVB, useDelta1, useDelta2, ↩
       VERBOSE )
270
271 if from < 1,
272     if VERBOSE,
273         warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣start␣index␣from␣', ...
274             num2str(from), '␣to␣1!']);
275     end
276     from = 1;
277 end
278 if to > length(signalENVB),
279     if VERBOSE,
280         warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
281             num2str(to), '␣to␣' num2str(length(signalENVB)), '!' ]);
282     end
283     to = length(signalENVB);
284 end
285 if useDelta1 && useDelta2,
286     out = signalENVB(from:to, :);
287 else
288     if useDelta1,
289         out = signalENVB(from:to, 1:16);
290     else
```

```
        if useDelta2,
            out = signalENVB(from:to, [1:8 17:24]);
        else
            out = signalENVB(from:to, 1:8);
        end
    end
end
end

% ==========================================================================
%%
function [out from to] = m_getMFCCChunk(from, to, signalMFCC, useDelta1, useDelta2,↩
    VERBOSE )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalMFCC),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalMFCC)), '!' ]);
    end
    to = length(signalMFCC);
end
if useDelta1 && useDelta2,
    out = signalMFCC(from:to, :);
else
    if useDelta1,
        out = signalMFCC(from:to, 1:16);
    else
        if useDelta2,
            out = signalMFCC(from:to, [1:8 17:24]);
        else
            out = signalMFCC(from:to, 1:8);
        end
    end
end
end
```

Listing B.58: functions/getTargetClass.m

```
function [uttTarget, uttPos, uttClz, clzLab, classLex] = getTargetClass(setup, seg,↩
    lex, classLex)

uttTarget=-1;
```

```
uttPos=-1;
uttClz=-1;
clzLab=[];
segSz = size(seg,1);
targetSz = length(setup.targets);
for tx=1:targetSz,
    t = setup.targets(tx);
    tpos = find(seg(:,3)==t,1);
    if ~isempty(tpos)
        uttTarget = t;
        uttPos    = tpos;
        break;
    end
end
if 0>uttTarget,
    return;
end
% Check classes: The vowel can only be at the first or last position
c1=0;
c2=0;
if uttPos==1,
    % VC(C) context
    if segSz==3,
        c1 = seg(2,3);
        c2 = seg(3,3);
    else
        c1 = seg(2,3);
    end
else
    % C(C)V context
    if segSz==3,
        c1 = seg(1,3);
        c2 = seg(2,3);
    else
        c1 = seg(1,3);
    end
end
for cx=1:length(setup.targetClasses)
    clz = setup.targetClasses{cx};
    if uttPos==1,
        if clz(1) == uttTarget,
            if length(clz) == 3,
                if clz(2) == c1 && clz(3) == c2,
                    uttClz = cx;
                    break;
                end
            else % length = 2
                if clz(2) == c1 && 0 == c2,
                    uttClz = cx;
```

```
                    break;
                end
            end
        end
    else
        if clz(end) == uttTarget,
            if length(clz) == 3,
                if clz(1) == c1 && clz(2) == c2,
                    uttClz = cx;
                    break;
                end
            else % length = 2
                if clz(1) == c1 && 0 == c2,
                    uttClz = cx;
                    break;
                end
            end
        end
    end
end
if 0>uttClz,
    uttTarget=-1;
    uttPos=-1;
else
    % Get class label for current target:
    if isempty(classLex{uttClz}),
        % construct label for this class:
        lab = '';
        clz = setup.targetClasses{uttClz};
        for x=1:length(clz),
            lab = strcat(lab, lex{clz(x)});
        end
        classLex{uttClz} = lab;
    end
    clzLab = classLex{uttClz};
end
end
```

Listing B.59: functions/getTargetClassC.m

```
function [uttTarget, uttPos, uttClz, clzLab, classLex] = getTargetClassC(setup, seg↩
    , lex, classLex)

uttTarget=-1;
uttPos=-1;
uttClz=-1;
clzLab=[];
segSz = size(seg,1);
targetSz = length(setup.targets);
```

```
for tx=1:targetSz,
    t = setup.targets(tx);
    tpos = find(seg(:,3)==t,1);
    if ~isempty(tpos)
        uttTarget = t;
        uttPos    = tpos;
        break;
    end
end
if 0>uttTarget,
    return;
end
% Check classes: The vowel can only be at the first or last position
c1=0;
c2=0;
if uttPos==1,
    % VC(C) context
    if segSz==3,
        c1 = seg(2,3);
        c2 = seg(3,3);
    else
        c1 = seg(2,3);
    end
else
    % C(C)V context
    if segSz==3,
        c1 = seg(1,3);
        c2 = seg(2,3);
    else
        c1 = seg(1,3);
    end
end
for cx=1:length(setup.targetClasses)
    clz = setup.targetClasses{cx};
    if uttPos==1,
        if clz(1) == uttTarget,
            if length(clz) == 3,
                g1 = clz(2);
                g2 = clz(3);
                if any(setup.targetGroups{g1}==c1) && any(setup.targetGroups{g2}==↩
                   c2),
                    uttClz = cx;
                    clzLab = [lex{uttTarget} 'CC'];
                    break;
                end
            else % length = 2
                g1 = clz(2);
                if any(setup.targetGroups{g1}==c1) && 0==c2 ,
                    uttClz = cx;
```

```
                    clzLab = [lex{uttTarget} 'C'];
                    break;
                end
            end
        end
    else
        if clz(end) == uttTarget,
            if length(clz) == 3,
                g1 = clz(1);
                g2 = clz(2);
                if any(setup.targetGroups{g1}==c1) && any(setup.targetGroups{g2}==↩
                   c2),
                    uttClz = cx;
                    clzLab = ['CC' lex{uttTarget}];
                    break;
                end
            else % length = 2
                g1 = clz(1);
                if any(setup.targetGroups{g1}==c1) && 0==c2,
                    uttClz = cx;
                    clzLab = ['C' lex{uttTarget}];
                    break;
                end
            end
        end
    end
end
if 0>uttClz,
    uttTarget=-1;
    uttPos=-1;
    uttClz=-1;
    clzLab=[];
else
    if isempty(classLex{uttClz}),
        % add label to lexicon
        classLex{uttClz} = clzLab;
    end
end
end
```

B.3. Context Sequence Model with articulatory data

The source code listings are ordered approximately according to the order in which the respective scripts and functions are called during the computer simulation runs.

B.3.1. Experiment with Polish EMA corpus

The discussion of this experiment can be found in chapter 6.3 starting on page 222; details about the corpus can be found in section A.1.

Listing B.60: runCSMEMA.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-10-03 based on: csmLeftOnly.m (2011-02-24)
%%%
%%% Eextension of the CSM production experiment with integration of
%%% articulatory data - using only the left context!
%%%
%%% Based on original code by Travis Wade.
%%%
%%% References:
%%%   Wade, T.; Dogil, G.; Schütze, H.; Walsh, M. & Möbius, B.:
%%%   "Syllable frequency effects in a context-sensitive segment production
%%%   model"; Journal of Phonetics, 2010, 38, 227-239.
%%%
%%%   Duran, D.; Schütze, H.; Walsh, M. & Möbius, B.:
%%%   "A computational model of unsupervised speech segmentation for
%%%   correspondence learning"; submitted to: Research on Language and
%%%   Computation, 2010.

THISMFILENAME = 'runCSMEMA';
%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('../io');
% ==============================================================================
% Initialize experiment setup
% ------------------------------------------------------------------------------
setup = struct([]);
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
setup.FsA = 16000; % sampling frequency of audio signal files
setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/csm+ema/';
setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/PolishEMAcorpus/';
setup.speakerDirs = {'JS1_emph/'; 'JS2_emph/'; 'NL_emph/'};% output
% ------------------------------------------------------------------------------
% CSM context settings:
% context size (in seconds)
setup.leftContextSec  = 0.5;
setup.rightContextSec = 0;
% use non-normalized data or either length-normalized vectors or z-scores:
setup.lengthnormalize = true;
setup.znormalize      = false;
setup.nuclei = {'a', 'y', 'i', 'u'};% PL only
setup.contextLabels = {'left', 'onset', 'nucl', 'coda', 'right'};% PL only
% ==============================================================================
```

```
% Audio representation
setup.useENVB   = true; % use amplitude envelope bands
setup.useMFCC   = false;% use MFCC
setup.useWAVE   = false;% (using the raw wave signal is not supported)
% Delta1 and Delta2 are the signals' first and second derivatives:
setup.useDelta1 = true;
setup.useDelta2 = true;
% the data types to use in the experiment:
% first column: articulatory data EMA
% second column: audio data ACC
% -> (1) EMA (2) ACC (3) EMA+ACC
setup.dataTypes = [1 0 ; 0 1 ; 1 1];
% print debugging info to console (and log file)
setup.VERBOSE=true;
setup.printProductionsToFile = true;
setup.EvalSylPos  = true;
setup.EvalSylType = true;
setup.EvalCA      = true; % context accuracy
setup.EvalCCA     = true; % context-class accuracy
% ---------------------------------------------------------------------------
% New: Initializations for phone class evaluation
% Using the same phone classes as for the experiments on MOCHA
%  lexPHON = 'p'  'r'  'a'  'l'  'k'  'i'  'y'  'u'
%             1    2    3    4    5    6    7    8
%
setup.phoneSort    = [5 1 2 4 6 7 3 8];% sorting of phones in output tables
setup.phoneClasses = {'PP'; 'LR'; 'VF'; 'VM'; 'VB'; 'XL'; 'XR'};
setup.phoneToClass = [1 2 4 2 1 3 4 5 6 7];
setup.SILCLASS = 6;
% The label number of SILENCE segments:
% Note: there is actually not "silence" label in the Polish EMA corpus, so
% this number just adds one to the number of existing eight labels.
setup.SIL = 9;
% Function handle for the function which reads the data from the
% pre-processed corpus files and create the memory sequence data structures
% used in the CSM simulation:
setup.memorySequenceFH = @createMemorySequence ;
setup.doSynthesis = false;
% ===========================================================================
% Check parameter consistency:
if ~ (setup.useENVB || setup.useMFCC),
    error('Must␣select␣at␣least␣one␣audio␣representation␣(ENV␣or␣MFCC)!');
end
% ---------------------------------------------------------------------------
% set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, 'leftOnly/', setup.timestamp, '/' ] ;
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
```

```
% ==============================================================================
%% run %%%
% Save terminal output to log file
diary(diaryFilename);
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp( '␣␣Setup:␣');
    disp(setup);
end
csmProduction( setup );
%% finished
disp(['␣Results␣stored␣to:␣', setup.baseDir]);
disp(['[', datestr(now,31), ']␣',THISMFILENAME, ':␣all␣done␣--␣start␣time␣was␣', ↩
   setup.timestamp, '.␣Bye!']);
diary off;
```

Listing B.61: runCSMEMAnoemph.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart

THISMFILENAME = 'runCSMEMAnoemph';
timestamp     = datestr(now,'yyyy-mm-dd+HH-MM');
%% Parameters:
% add folders with required matlab files to the search path:
addpath('functions');
addpath('../io');
% ==============================================================================
% Initialize setup
% ------------------------------------------------------------------------------
setup = struct([]);
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
setup.FsA = 16000; % sampling frequency of audio signal files
setup.OUTDIR = '~/a2/duran_working/diss/csm+ema/';
setup.matlabDir = '~/a2/duran_working/diss/db/PolishEMAcorpus/';
setup.speakerDirs = {'JS1_noemph/'; 'JS2_noemph/'; 'NL_noemph/'};% output
% context size (in seconds)
setup.leftContextSec  = 0.5;
setup.rightContextSec = 0;
% use non-normalized data or either length-normalized vectors or z-scores:
setup.lengthnormalize = true;
setup.znormalize      = false;
setup.nuclei = {'a', 'y', 'i', 'u'};
setup.contextLabels = {'left', 'onset', 'nucl', 'coda', 'right'};
setup.printProductionsToFile = false;
setup.EvalSylPos  = true;
setup.EvalSylType = true;
setup.EvalCA      = true; % context accuracy
setup.EvalCCA     = true; % context-class accuracy
```

```
% ==============================================================================
% Audio representation
setup.useENVB   = false;% use amplitude envelope bands
setup.useMFCC   = true;% use MFCC
setup.useWAVE   = false; % using the raw wave signal not supported
% Delta1 and Delta2 are the signals' first and second derivatives:
setup.useDelta1 = true;
setup.useDelta2 = true;
% the data types to use in the experiment:
% first column: articulatory data EMA
% second column: audio data ACC
% -> (1) EMA (2) ACC (3) EMA+ACC
setup.dataTypes = [ 1 0 ; 0 1 ; 1 1];
% print debugging info to console (and log file)
setup.VERBOSE=true;
% ------------------------------------------------------------------------------
% Initializations for phone class evaluation
setup.phoneSort    = [5 1 2 4 6 7 3 8];% sorting of phones in output tables
setup.phoneClasses = {'PP'; 'LR'; 'VF'; 'VM'; 'VB'; 'XL'; 'XR'};
setup.phoneToClass = [1 2 4 2 1 3 4 5 6 7];
setup.SIL = 9;
setup.SILCLASS = 6;
% Function handle for the function which reads the data from the
% pre-processed corpus files and create the memory sequence data structures
% used in the CSM simulation:
setup.memorySequenceFH = @createMemorySequence ;
setup.doSynthesis = false;
% ==============================================================================
% Check parameter consistency:
if ~ (setup.useENVB || setup.useMFCC),
    error('Must␣select␣at␣least␣one␣audio␣representation␣(ENV␣or␣MFCC)!');
end
% ------------------------------------------------------------------------------
% set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, 'leftOnly/', timestamp, '/' ] ;
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
% ==============================================================================
%% run %%%
% Save terminal output to log file
diary(diaryFilename);
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']␣␣Start:␣', THISMFILENAME ]);
    disp(['␣␣Saving␣diray␣to:␣', diaryFilename]);
    disp( '␣␣Setup:␣');
    disp(setup);
end
csmProduction( setup );
```

```
%% finished
disp([' Results stored to: ', setup.baseDir]);
disp(['[', datestr(now,31), '] ',THISMFILENAME, ': all done - start time was ', timestamp, '. Bye!']);
diary off;
```

Listing B.62: functions/csmProduction.m

```
function csmProduction( setup )
%CSMPRODUCTION Run CSM segment production

% Loop through all speaker sub-corpora:
for speakerNr = 1:length(setup.speakerDirs),
    % Working parameters (autosettings):
    speaker =  setup.speakerDirs{speakerNr}(1:end-1) ;
    if setup.VERBOSE,
        disp(['== Start speaker ' speaker ' ==']);
    end
    % Loop through all data type conditions:
    for dT = 1:size(setup.dataTypes,1),
        useART = setup.dataTypes(dT,1);% use articulatory signal data?
        useACC = setup.dataTypes(dT,2);% use acoustic signal data?
        % Initialize base name for the current speaker and data type and
        % create output directory:
        baseFilename = initOutputBase(setup, speakerNr, useART, useACC);
        % == Initializations for current speaker and data type ============
        load([setup.matlabDir setup.speakerDirs{speakerNr} 'index.mat'],'corpusFs');
        if setup.VERBOSE,
            fprintf('corpusFs=%dHz\n', corpusFs);
        end
        % Set left and right context size in number of frames according to the
        % specified context lengths in seconds and the frame rate of the corpus
        % data:
        setup.lc = ceil(setup.leftContextSec  * corpusFs);
        setup.rc = ceil(setup.rightContextSec * corpusFs);
        % The expected data structures are:
        %
        % corpus    (N x D) with N = number of frames,
        %            D = number of dimensions
        %
        % labels    (N x 7) -- columns
        %            [:,1] word label index number
        %            [:,2] phon label index number
        %            [:,3] EMA:lip distance label
        %            [:,4] EMA:tongue body 1 label
        %            [:,5] EMA:tongue tip label
        %            [:,6] file index
        %            [:,7] segment index
```

```
41          %
42          % segments  (S x 5) with columns:
43          %            [:,1] start index in corpus
44          %            [:,2] end index in corpus
45          %            [:,3] label = -1 for unlabeled left,
46          %                           0 for unlabeld right
47          %                             context, otherwise phone label index
48          %                             number
49          %            [:,4] file index number
50          %            [:,5] context = {-1,1,2,3,0}, with:
51          %                    1 = left context (preceding syllable/silence)
52          %                    2 = onset position in syllable
53          %                    3 = nucleus position
54          %                    4 = coda position
55          %                    5 = right context (next syllable/silence)
56          %            [:,6] phone class
57          %
58          %   files   (F x 1) matrix
59          %
60          %   rightContext [] -- not used in current implementation
61          %
62          %   lex     { x } cell array -- phon labels lexicon
63          %            the phone label index numbercorresponds to a label
64          %            string in this cell array
65          %
66          %   targets (S x 1) matrix specifying for each segment if it is a
67          %            production target (1) or not (0)
68          %
69          [ corpus, labels, segments, files, rightContext, lex, targets ] = ...
70              setup.memorySequenceFH(setup, speakerNr, useACC, useART);
71          % "segments" contains all production target as well as
72          % pseudo-segments which represent the left and right contexts of
73          % the very first and very last target segments in each utterance
74          % (depending on current setup)
75          % DEBUG
76          producedTargets = zeros(size(targets,1),1);
77          % initialize output:
78          % productions = (T x 9), with T=number of targets
79          %  column 1: file
80          %  column 2: label
81          %  column 3: original target segment number
82          %  column 4: production segment number
83          %  column 5: start frame in productionFrames
84          %  column 6: end frame in productionFrames
85          %  column 7: c-match value
86          %  column 8: left match value
87          %  column 9: rigth match value
88          productions = zeros(sum(targets), 9);
89          % rows:     corpus frames
```

```
90          % column 1: original frame index
91          productionFrames = zeros(size(corpus,1),1);
92          if setup.EvalCA,
93              %%% Initialize matrices for the evaluation
94              % Matrix with statistics for each produced segment about:
95              %  column 1: The total number of available candidate exemplars
96              %            (segments with the same label in the remaining corpus
97              %            without the original target utterance)
98              %  column 2: The number of available correct left segmental
99              %            contexts
100             %            (candidate segments with a preceding segment which
101             %            labels matches that of the targets' preceding
102             %            segment)
103             %  column 3: The number of available matching right contexts
104             %  column 4: The number of available matching left and right
105             %            segmental contexts.
106             evalBaselineSegments = NaN(size(productions,1), 4);
107             evalBaselineClasses  = NaN(size(productions,1), 4);
108         else
109             evalBaselineSegments = [];
110             evalBaselineClasses  = [];
111         end
112         % --- run utterance production experiment -------------------------
113         %%% go through the list of production targets
114         p = 0;  % production segment counter
115         pf = 1; % production frame counter
116         for t = 1:size(segments,1),
117             thisFile  = segments(t,4);
118             thisLabel = segments(t,3);
119             if setup.VERBOSE,
120                 % print information about the current segment to
121                 % the screen.
122                 debugLabel = '';
123                 if thisLabel > 0,
124                     % 0 is not a valid matrix index in Matlab, but it is
125                     % used here to mark unlabeld parts of the corpus.
126                     debugLabel = lex{thisLabel};
127                 end
128                 if segments(t,5)>0,
129                     fprintf('NEXT:␣file␣%3d;␣seg.%5d;␣[%2d:%3s](%5s);␣target=%d\n',↩
                            ...
130                         thisFile, t, thisLabel, debugLabel, ...
131                         setup.contextLabels{segments(t,5)}, targets(t) );
132                 else
133                     fprintf('NEXT:␣file␣%3d;␣seg.%5d;␣[%2d:%3s];␣target=%d\n', ...
134                         thisFile, t, thisLabel, debugLabel, targets(t));
135                 end
136                 clear debugLabel ;
137             end
```

```
138             if thisLabel == -1,
139                 % The current segment is preceding the first labeld segment
140                 % in an utterance:
141                 % this is a "left context" -> add signal data from original
142                 % corpus to production output
143                 if setup.VERBOSE,
144                     disp(['Adding␣original␣left␣context␣from␣file␣', num2str(↩
                           thisFile) ]);
145                 end
146                 thisLC = segments(t,2) - segments(t,1) + 1;
147                 if thisLC < setup.lc,
148                     % The available left context is too short
149                     warning('DD:UnexpectedValue',['Left␣context␣=␣' num2str(thisLC)↩
                           ]);
150                 end
151                 productionFrames(pf:pf+thisLC-1) = segments(t,1) : segments(t,2) ;
152                 pf = pf+thisLC;
153                 clear thisLC;
154             else
155                 if thisLabel==0,
156                     % the current segment is following the last one in an
157                     % utterance:
158                     % "right context" -> production finished
159                     if setup.VERBOSE,
160                         disp(['Finished␣production␣of␣utterance␣from␣file␣', ↩
                               num2str(thisFile) ]);
161                     end
162                     % (the right context is not considered in this
163                     % implementation, so this block does essentially
164                     % nothing but printing some info to the log)
165                 else
166                     % The current segment is an actual target segment
167                     % produce segment which matches thisLabel
168                     % get the left context for the current target:
169                     % this is always a window of a fixed length immediately
170                     % preceding the current production target; i.e. take
171                     % the last LC frames from the output:
172                     pIndex = productionFrames(pf-setup.lc : pf-1);
173                     refContextLeft  = corpus( pIndex, :);
174
175                     % NOT USING RIGHT CONTEXT:
176                     %refContextRight = getRightContextVector( t, rightContext, size↩
                           (lex,1) ) ;
177
178                     % == Get cloud of candidate exemplars ==
179                     % The structure of "cloud" is identical to the
180                     % structure of "segments" but it contains only those
181                     % segments, which do not belong to the current
182                     % utterance (i.e. which are not from the same file),
```

```
183                 % and which have the same label as the current
184                 % procudtion target (i.e. which are actual candidates
185                 % for production).
186                 cloud = segments( segments(:,4) ~=thisFile & segments(:,3)==↩
                       thisLabel, :);
187                 % initialize matrix for the scores:
188                 cloudCmatchScores = zeros(size(cloud,1),3);
189                 if setup.EvalCA,
190                     % ==========================================
191                     % Initialize matrices for the baseline evaluation
192                     % of this target
193                     thisEvalBaselineSegment    = zeros(1,4);
194                     thisEvalBaselineSegment(1) = size(cloud,1);
195                 end
196                 clzCloud = segments( segments(:,4) ~=thisFile & segments(:,6)==↩
                       setup.phoneToClass(thisLabel), :);
197                 thisEvalBaselineClass    = zeros(1,4);
198                 thisEvalBaselineClass(1) = size(clzCloud,1);
199                 clear clzCloud;
200                 if 0 == thisEvalBaselineClass(1),
201                     error('DD:EmptySet', 'Empty␣cloud␣found!');
202                     % This should not happen!
203                 end
204                 %%% go through all the candidates
205                 for c=1:size(cloud,1),
206                     candStart = cloud(c,1) ;
207                     candContextLeft = corpus( candStart-setup.lc : candStart-1,↩
                           :) ;
208                     % Save statistics to compute the base line
209                     % probability (i.e. selection of a "correct"
210                     % candidate by chance)
211                     if setup.EvalCA,
212                         % Get the segment index of the candidate exemplar:
213                         candx = labels(candStart,7);
214                         [thisEvalBaselineSegment, thisEvalBaselineClass] = ...
215                             countBaselineStats(setup, segments, candx, t, ...
216                             thisEvalBaselineSegment, thisEvalBaselineClass) ;
217                     end
218                     % Compute c-match values for the current production
219                     % context and the context of the current candidate
220                     % segment (including left and right half-contexts)
221                     [match lmatch rmatch] = cmatch( refContextLeft, ↩
                           candContextLeft, 0, 0 ) ;
222                     % Store results for the current candidate:
223                     cloudCmatchScores(c,:) = [match lmatch rmatch];
224                 end %c
225                 [val idx] = max(cloudCmatchScores(:,1));
226                 best = cloud(idx,:);
227                 lval = cloudCmatchScores(idx,2);
```

```
                    rval = cloudCmatchScores(idx,3);
                    % now add the winner to the production sequence:
                    bestSegnum = find(segments(:,1) == best(1) ) ;
                    bestLen    = best(2) - best(1) +1;
                    p = p+1;
                    productions(p,:) = [thisFile, thisLabel, t, bestSegnum, pf, pf+↩
                       bestLen-1, val, lval, rval];
                    productionFrames(pf:pf+bestLen-1) = segments(bestSegnum,1) : ↩
                       segments(bestSegnum,2) ;
                    pf = pf+bestLen;
                    producedTargets(t) = 1;
                    if setup.EvalCA,
                        evalBaselineSegments(p,:) = thisEvalBaselineSegment;
                        evalBaselineClasses(p,:)  = thisEvalBaselineClass;
                    end
                end
            end
        end % t
        if p < size(productions,1),
            warning('DD:UnexpectedValue',['Estimated␣number␣of␣productions=' ...
                num2str(size(targets,1)), ';␣produced=' ...
                num2str(p),  '.␣Truncating␣matrices!']);

            productions = productions(1:p,:);
            if setup.EvalCA,
                evalBaselineSegments = evalBaselineSegments(1:p,:);
                evalBaselineClasses = evalBaselineClasses(1:p,:);
            end
        end
        % ================================================================
        % Run evaluation
        evaluateCSM( setup, speakerNr, dT, lex, segments, productions, ...
            evalBaselineSegments, evalBaselineClasses ) ;
        % ================================================================
        % Produce audio file from model productions:
        if setup.doSynthesis,
            synthesizeWave( setup, speakerNr, productions, segments, ...
                files,  baseFilename, setup.VERBOSE );
        end
    end %FOR dT (data type)
    disp(['==␣Finished␣speaker␣' speaker '␣==']);
end % FOR speakerNr
end
```

Listing B.63: functions/initOutputBase.m

```
function baseFilename = initOutputBase( setup, speakerNr, useART, useACC )
%initOutputBase : Initialize output file base-name and create directory

```

```
speaker =  setup.speakerDirs{speakerNr}(1:end-1) ;
% Initialize base name for the current speaker and data type:
baseFilename = [setup.baseDir, speaker, '/'] ;
if useART,
    baseFilename = [baseFilename 'EMA_'];
end
if useACC,
    if setup.useMFCC,
        baseFilename = [baseFilename 'MFCC_'];
    end
    if setup.useENVB,
        baseFilename = [baseFilename 'ENV_'];
    end
end
if setup.useDelta1,
    baseFilename = [baseFilename 'd1_'];
end
if setup.useDelta2,
    baseFilename = [baseFilename 'd2_'];
end
baseFilename  = [baseFilename(1:end-1), '/' ];
% create output directory for current speaker and data type:
if ~exist(baseFilename,'dir'),
    mkdir(baseFilename);
end
end
```

Listing B.64: functions/createMemorySequence.m

```
function [ corpus, labels, segments, files, rightContext, lex, targets ] = ...
    createMemorySequence( setup, speakerNr, useACC, useART )
%createMemorySequence : Create a concatenated memory sequence from corpus.
%
%   Create memory sequence according to the specified input parameters
%
% IN:  setup         struct (see: runCSMEMA.m)
%      speakerNr     int {1,2,3}
%      useACC        boolean flag: use acoustic signal data?
%      useART        boolean flag: use articulatory signal data?
%
% OUT: corpus        (N x D) with N = number of frames,
%                    D = number of dimensions
%      labels        (N x 6) - columns
%                      [:,1] word label index number
%                      [:,2] phon label index number
%                      [:,3] EMA:lip distance label
%                      [:,4] EMA:tongue body 1 label
%                      [:,5] EMA:tongue tip label
%                      [:,6] file index
```

```
21 %      segments      optional:
22 %                    (S x 5), with columns:
23 %                      [:,1] start
24 %                      [:,2] end
25 %                      [:,3] label = -1 for left, 0 for right context,
26 %                                    otherwise label index number
27 %                      [:,4] file
28 %                      [:,5] context = {-1,1,2,3,0}, with:
29 %                            1 = left context (preceding syllable/silence)
30 %                            2 = onset position in syllable
31 %                            3 = nucleus position
32 %                            4 = coda position
33 %                            5 = right context (next syllable/silence)
34 %                      [:,6] phone class
35 %
36 %      files         (F x 1)
37 %      rightContext  (S x 2)
38 %                      rows: segments
39 %                      columns 1 and 2: the right context labels
40 %      lex           { x } cell array
41 %
42 %      targets   (S x 1) matrix
43
44 %%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
45 %%% file created: 2011-01-28
46
47 rightContext=[];
48 lex={};
49 targets=[];
50 % == check input arguments ==
51 if setup.lengthnormalize && setup.znormalize,
52     disp(['[', datestr(now,31), '␣', mfilename, ']␣' ...
53         'Cannot␣select␣both␣lengthnormalize␣and␣znormalize!␣Setting␣'...
54         'znormalize=false']);
55     setup.znormalize = false;
56 end
57 % == check output arguments ==
58 if nargout == 7,
59     doRightContext = true;
60     doSeg          = true;
61 else
62     doRightContext = false;
63     if nargout < 3,
64         doSeg = false;
65     else
66         doSeg = true;
67     end
68 end
69 % ------------------------------------------------------------------------
```

```
% load index and label data for this speaker:
inDir = [setup.matlabDir setup.speakerDirs{speakerNr}];
load([inDir 'index.mat']);
% check whether the variables are loaded successfully
if ~exist('baseNames','var') || ...
        ~exist('totalFramesEma','var') || ~exist('totalFramesWav','var') || ~exist('totalFramesEnv','var') || ~exist('totalFramesMFCC','var') ||  ~exist('segmentsPHON','var') || ~exist('segmentsWORD','var') || ~exist('corpusFs','var'),
    % add more checks?
    error('index.mat was not opened successfully!');
end
% ----------------------------------------------------------------------
offset=0;
accCols = [];
emaCols = [];
envCols = [];
mfcCols = [];
if useACC,
    if setup.useWAVE,
        disp('WARNING: adding WAVE not implemented!');
    end
    if setup.useMFCC,
        plus = 13;
        if setup.useDelta1 && setup.useDelta2,
            plus=39;
        else
            if setup.useDelta1 || setup.useDelta2,
                plus=26;
            end
        end
        mfcCols = (offset+1) : (offset+plus);% the MFCC columns in "corpus"
        offset = offset + plus;
    end
    if setup.useENVB,
        % use amplitude envelop representation (default for audio)
        plus = 8;
        if setup.useDelta1 && setup.useDelta2,
            plus=24;
        else
            if setup.useDelta1 || setup.useDelta2,
                plus=16;
            end
        end
        envCols = (offset+1) : (offset+plus);% the envelop columns in "corpus"
        offset = offset + plus;
    end
    accCols = [mfcCols envCols];
end
```

```
if useART,
    plus = 8;
    if setup.useDelta1 && setup.useDelta2,
        plus=24;
    else
        if setup.useDelta1 || setup.useDelta2,
            plus=16;
        end
    end
    emaCols = (offset+1) : (offset+plus);
    offset = offset + plus;
end
% ----------------------------------------------------------------------
% initialize output data
rows   = max([totalFramesEma, totalFramesEnv, totalFramesMFCC]) ;
corpus = zeros(rows, offset);
labels = zeros(rows,7);% columns: word, phon, tbo1, ttip, llip, file, seg.no.
if doSeg,
    % start \t end \t label \t file
    segments = zeros(rows, 6);
    files = zeros( size(baseNames,1) , 1);
end
if setup.VERBOSE,
    disp(['[', datestr(now,31), '␣', mfilename, ']␣initialized␣corpus␣with␣'...
        num2str(rows), '␣rows␣and␣', num2str(offset), '␣columns' ]);
end
ptrAll    = 1;
ptrAllEnd = 1;
scnt = 0;% segment counter
fcnt = 0;% file counter
% loop through utterances
for fdx=1:length(baseNames),
    bfile = baseNames{fdx};
    skip  = false;
    % the segments of this utterance:
    seg = segmentsPHON( segmentsPHON(:,4)==fdx , :);
    labelsDone = false;
    ptrEmaEnd = 0;
    ptrEnvEnd = 0;
    %%% Check whether all data is available
    if useART,
        if sum(checkFiles(fdx,1:4)) ~= 4,
            if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣EMA␣↩
               data']); end
            skip = true ;
        end
    end
    if useACC || setup.useWAVE || setup.useMFCC,
        if checkFiles(fdx,5) ~= 1,
```

```
            if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣Audio␣data']); end
            skip = true ;
        end
    end
    if skip,
        continue;
    end
    % == EMA ======================================================== EMA ==
    if useART,
        % set pointer
        ptrEma = ptrAll;
        % load EMA data for the current utterance:
        load([inDir bfile '_ema.mat']);
        load([inDir bfile '_frames.mat']);% reads from file: "frames"
        if setup.lc > 0,
            % get left context from signal file
            % (stretch of data preceding the fist segment)
            from = seg(1,1) - setup.lc;
            to   = seg(1,1) - 1;
            [lc from to] = m_getChunk(from, to, setup.VERBOSE, setup.useDelta1, setup.useDelta2, ...
                signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
                signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy );
            ptrEmaEnd = ptrEma+size(lc,1)-1;
            corpus(ptrEma:ptrEmaEnd, emaCols) = lc;
            labels(ptrEma:ptrEmaEnd,6)   = repmat( fdx, size(lc,1), 1);
            labels(ptrEma:ptrEmaEnd,1:5) = frames(from:to,:);
            if doSeg,
                scnt = scnt +1;
                segments(scnt, : ) = [ptrEma ptrEmaEnd -1 fdx 1 setup.SILCLASS];
                labels(ptrEma:ptrEmaEnd,7)   = repmat( scnt, size(lc,1), 1);
            end
            clear lc ;
        end
        % get segments:
        context = 2; % 2=syllable onset
        for sdx = 1:size(seg,1),
            from = seg(sdx,1) ;
            to   = seg(sdx,2) ;
            [sdat from to] = m_getChunk(from, to, setup.VERBOSE, setup.useDelta1, setup.useDelta2, ...
                signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
                signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy );
            ptrEma = ptrEmaEnd +1 ;
            ptrEmaEnd = ptrEma+size(sdat,1)-1;
            corpus(ptrEma:ptrEmaEnd, emaCols) = sdat;
            labels(ptrEma:ptrEmaEnd,6)   = repmat( fdx, size(sdat,1), 1);
            labels(ptrEma:ptrEmaEnd,1:5) = frames(from:to,:);
```

```
            if doSeg,
                scnt = scnt +1;
                lab = lexPHON{seg(sdx,3)};
                clz = setup.phoneToClass(seg(sdx,3));
                if find(strcmp(lab, setup.nuclei)) ~= 0,
                    context = 3; % 3=syllable nucleus
                else if context==3,
                        % we've already seen the nucleus -> now: coda
                        context = 4;% 4=syllable coda
                    end
                end
                segments(scnt, : ) = [ptrEma ptrEmaEnd seg(sdx,3) fdx context clz];
                labels(ptrEma:ptrEmaEnd,7)   = repmat( scnt, size(sdat,1), 1);
            end
        end % for sdx
        % get right context:
        if setup.rc>0,
            from = seg(end,2) +1 ;
            to   = seg(end,2) + setup.rc;
            [rc from to] = m_getChunk(from, to, setup.VERBOSE, setup.useDelta1, ↩
               setup.useDelta2, ...
                signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
                signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy );
            ptrEma = ptrEmaEnd +1 ;
            ptrEmaEnd = ptrEma+size(rc,1)-1;
            corpus(ptrEma:ptrEmaEnd, emaCols) = rc;
            %labels(ptrEma:ptrEmaEnd,2) = repmat( fdx, size(rc,1), 1);
            labels(ptrEma:ptrEmaEnd,6)   = repmat( fdx, size(rc,1), 1);
            if to > size(frames,1),
                d = to - size(frames,1);
                toF = to - d;
                pF = ptrEmaEnd - d;
            else
                toF = to;
                pF = ptrEmaEnd;
            end
            labels(ptrEma:pF,1:5) = frames(from:toF,:);

            if doSeg,
                scnt = scnt +1;
                segments(scnt, : ) = [ptrEma ptrEmaEnd 0 fdx 5 setup.SILCLASS];% 5=↩
                   right context

            end
        end
        if setup.VERBOSE,
            disp(['[', datestr(now,31), ']_finished_EMA_signal_from:_', bfile]);
        end
        labelsDone = true;
```

```
    end % useART
    % == ENV ======================================================== ENV ==
    if useACC,
        % set pointer
        ptrEnv = ptrAll;
        % load ENV data for the current utterance:
        file = [inDir bfile '_audio.mat'];
        load(file);
        framesFile = [inDir bfile '_frames.mat'];
        load(framesFile);
        % add left context
        if setup.lc > 0,
            from = seg(1,1) - setup.lc;
            to   = seg(1,1) - 1;
            if setup.useMFCC,
                [lcM from to] = priv_getMFCCChunk(from, to, signalMFCC, ...
                    setup.useDelta1, setup.useDelta2, setup.VERBOSE );
            else
                lcM=[];
            end
            if setup.useENVB,
                [lcE from to] = priv_getEnvChunk(from, to, signalENVB, ...
                    setup.useDelta1, setup.useDelta2, setup.VERBOSE );
            else
                lcE=[];
            end
            lc = [lcM lcE];
            ptrEnvEnd = ptrEnv+size(lc,1)-1;
            corpus(ptrEnv:ptrEnvEnd, accCols) = lc;
            if ~labelsDone,
                labels(ptrEnv:ptrEnvEnd,6)   = repmat( fdx, size(lc,1), 1);
                labels(ptrEnv:ptrEnvEnd,1:5) = frames(from:to,:);

                if doSeg,
                    scnt = scnt +1;
                    segments(scnt, : ) = [ptrEnv ptrEnvEnd -1 fdx 1 setup.SILCLASS↩
                        ];
                    labels(ptrEnv:ptrEnvEnd,7)   = repmat(scnt, size(lc,1), 1);
                end
            end
            clear lc ;
        else
            %ptrEmaEnd = 0;
        end
        % get segments:
        context = 2; % 2=syllable onset
        for sdx = 1:size(seg,1),
            from = seg(sdx,1) ;
            to   = seg(sdx,2) ;
```

```
305             if setup.useMFCC,
306                 [sdatM from to] = priv_getMFCCChunk(from, to, signalMFCC, ...
307                     setup.useDelta1, setup.useDelta2, setup.VERBOSE );
308             else
309                 sdatM=[];
310             end
311             if setup.useENVB,
312                 [sdatE from to] = priv_getEnvChunk(from, to, signalENVB, ...
313                     setup.useDelta1, setup.useDelta2, setup.VERBOSE );
314             else
315                 sdatE=[];
316             end
317             sdat = [sdatM sdatE];
318             ptrEnv = ptrEnvEnd +1 ;
319             ptrEnvEnd = ptrEnv+size(sdat,1)-1;
320             corpus(ptrEnv:ptrEnvEnd, accCols) = sdat;
321             if ~labelsDone,
322                 labels(ptrEnv:ptrEnvEnd,6)   = repmat( fdx, size(sdat,1), 1);
323                 labels(ptrEnv:ptrEnvEnd,1:5) = frames(from:to,:);
324                 if doSeg,
325                     scnt = scnt +1;
326                     lab = lexPHON{seg(sdx,3)};
327                     clz = setup.phoneToClass(seg(sdx,3));
328                     if find(strcmp(lab, setup.nuclei)) ~= 0,
329                         context = 3;
330                     else if context==3,
331                             % we've already seen the nucleus -> now: coda
332                             context = 4;
333                         end
334                     end
335                     segments(scnt, : ) = [ptrEnv ptrEnvEnd seg(sdx,3) fdx context ↩
                           clz];
336                     labels(ptrEnv:ptrEnvEnd,7)   = repmat(scnt, size(sdat,1), 1);
337                 end
338             end
339         end % for sdx
340         % get right context:
341         if setup.rc>0,
342             from = seg(end,2) +1 ;
343             to   = seg(end,2) + setup.rc;
344             if setup.useMFCC,
345                 [rcM from to] = priv_getMFCCChunk(from, to, signalMFCC, ...
346                     setup.useDelta1, setup.useDelta2, setup.VERBOSE );
347             else
348                 rcM=[];
349             end
350             if setup.useENVB,
351                 [rcE from to] = priv_getEnvChunk(from, to, signalENVB, ...
352                     setup.useDelta1, setup.useDelta2, setup.VERBOSE );
```

```
353             else
354                 rcE=[];
355             end
356             rc = [rcM rcE];
357             ptrEnv = ptrEnvEnd +1 ;
358             ptrEnvEnd = ptrEnv+size(rc,1)-1;
359             corpus(ptrEnv:ptrEnvEnd, accCols) = rc;
360             if ~labelsDone,
361                 labels(ptrEnv:ptrEnvEnd,6)  = repmat( fdx, size(rc,1), 1);
362                 if to > size(frames,1),
363                     d = to - size(frames,1);
364                     toF = to - d;
365                     pF = ptrEnvEnd - d;
366                 else
367                     toF = to;
368                     pF = ptrEnvEnd;
369                 end
370                 labels(ptrEnv:pF,1:5) = frames(from:toF,:);
371
372                 if doSeg,
373                     scnt = scnt +1;
374                     segments(scnt, : ) = [ptrEnv ptrEnvEnd 0 fdx 5 setup.SILCLASS];
375
376                 end
377             end
378         end
379         if setup.VERBOSE, disp(['[', datestr(now,31), ']␣finished␣ENV␣signal␣from:␣↩
               ', bfile]); end
380         clear rc
381         labelsDone = true;
382         %%% %%%
383     end % useACC
384     % ==================================================================
385     %%% current file ("utterance") done
386     if doSeg,
387         fcnt = fcnt +1;
388         files(fcnt) = fdx;
389     end
390     ptrAllEnd = max([ptrEmaEnd ptrEnvEnd]);
391     ptrAll = ptrAllEnd+1;
392 end % for fdx
393 %%% trim output data
394 if setup.VERBOSE,
395     disp(['[', datestr(now,31), ']␣trimming␣data␣from␣', num2str(size(corpus,1)), ↩
           ...
396         '␣rows␣down␣to␣', num2str(ptrAllEnd) ]);
397 end
398 corpus = corpus(1:ptrAllEnd,:);
399 labels = labels(1:ptrAllEnd,:);
```

```
if doSeg,
    if setup.VERBOSE,
        disp(['[', datestr(now,31), ']␣trimming␣segment␣data␣from␣', num2str(size(segments,1)), ...
            '␣rows␣down␣to␣', num2str(scnt) ]);
    end
    segments = segments(1:scnt,:);
    if setup.VERBOSE,
        disp(['[', datestr(now,31), ']␣trimming␣file␣index␣from␣', num2str(size(files,1)), ...
            '␣rows␣down␣to␣', num2str(fcnt) ]);
    end
    files = files(1:fcnt,:);
    %%% add right context labels
    if doRightContext,
        disp('**␣Adding␣right-context␣labels␣**');
        rightContext = zeros(size(segments,1),2);
        lex = lexPHON'; %'
        % for all files...
        for fdx = 1:size(files,1),
            thisFile = files(fdx);
            segmentsFile = find(segments(:,4)==thisFile);
            % new file: get right context:
            wx= segmentsWORD(:,4)==thisFile;
            thisWord = segmentsWORD(wx,3);
            if isempty(thisWord) || thisWord < 1 || thisWord > 16,
                disp('ERROR');
            end
            [ rcLabels ] = getTranscription( lexWORD, thisWord );
            %%% add segment labels:
            b = find( strcmp(lex, rcLabels{1}), 1);
            if isempty(b)
                lex = [lex; rcLabels{1}];
                b = size(lex,1);
            end
            c = find( strcmp(lex, rcLabels{2}), 1);
            if isempty(c)
                lex = [lex; rcLabels{2}];
                c = size(lex,1);
            end
            for x = size(segmentsFile,1)-1:-1:2,

                segIndex = segmentsFile(x);
                % assign context:
                rightContext(segIndex,:) = [b c];
                % shift labels:
                c = b;
                b = segments(segIndex,3);
            end
```

```
447         end % fdx
448     end % doRightContext
449 end
450 %%% normalize output data
451 if setup.lengthnormalize,
452     if setup.VERBOSE, disp(['[', datestr(now,31), ']␣length-normalizing␣data...']);↩
            end
453     outDataNorm = zeros(size(corpus));
454     for idx = 1:size(corpus,1) ,
455         if useACC,
456             if setup.useMFCC,
457                 n = norm(corpus(idx,mfcCols));
458                 nvec = corpus(idx,mfcCols) ./ n ;
459                 outDataNorm(idx,mfcCols) = nvec;
460             end
461             if setup.useENVB,
462                 n = norm(corpus(idx,envCols));
463                 nvec = corpus(idx,envCols) ./ n ;
464                 outDataNorm(idx,envCols) = nvec;
465             end
466         end
467         if useART,
468             n = norm(corpus(idx,emaCols));
469             nvec = corpus(idx,emaCols) ./ n ;
470             outDataNorm(idx,emaCols) = nvec;
471         end
472         n = norm(outDataNorm(idx,:));
473         nvec = outDataNorm(idx,:) ./ n ;
474         outDataNorm(idx,:) = nvec;
475     end
476     corpus =outDataNorm;
477 end
478 %%% create production targets:  (all segments which are labeled)
479 targets = zeros(size(segments,1),1);
480 targets(  segments(:,3)> 0  ) = 1;
481 if setup.VERBOSE,
482     disp(['[', datestr(now,31), '␣', mfilename, ']␣␣finished␣creating␣memory␣↩
           sequence']);
483 end
484 end
485 % ==========================================================================
486 %% get chunk from the signal in the range [from:to]
487 % added 2012-08-28: use both x and y EMA signals
488 function [out from to] = m_getChunk(from, to, VERBOSE, useDelta1, useDelta2, ...
489     signalLLIPx, signalLLIPy, signalTBO1x, signalTBO1y, ...
490     signalTBO2x, signalTBO2y, signalTTIPx, signalTTIPy )
491
492 if from < 1,
493     if VERBOSE,
```

```
        warning('DD:INDEXOUTOFRANGE', ['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalLLIPx),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalLLIPx)), '!' ]);
    end
    to = length(signalLLIPx);
end
if useDelta1 && useDelta2,
    dims = 24;
else
    if useDelta1 || useDelta2,
        dims = 16;
    else
        dims = 8;
    end
end
out = zeros(to - from +1, dims);
out(:,1) = signalLLIPx(from:to,1);
out(:,2) = signalLLIPy(from:to,1);
out(:,3) = signalTBO1x(from:to,1);
out(:,4) = signalTBO1y(from:to,1);
out(:,5) = signalTBO2x(from:to,1);
out(:,6) = signalTBO2y(from:to,1);
out(:,7) = signalTTIPx(from:to,1);
out(:,8) = signalTTIPy(from:to,1);
if useDelta1,
    out(:,9)  = signalLLIPx(from:to,2);
    out(:,10) = signalLLIPy(from:to,2);
    out(:,11) = signalTBO1x(from:to,2);
    out(:,12) = signalTBO1y(from:to,2);
    out(:,13) = signalTBO2x(from:to,2);
    out(:,14) = signalTBO2y(from:to,2);
    out(:,15) = signalTTIPx(from:to,2);
    out(:,16) = signalTTIPy(from:to,2);
end
if useDelta2,
    out(:,17) = signalLLIPx(from:to,3);
    out(:,18) = signalLLIPy(from:to,3);
    out(:,19) = signalTBO1x(from:to,3);
    out(:,20) = signalTBO1y(from:to,3);
    out(:,21) = signalTBO2x(from:to,3);
    out(:,22) = signalTBO2y(from:to,3);
    out(:,23) = signalTTIPx(from:to,3);
    out(:,24) = signalTTIPy(from:to,3);
```

```
end
end
% ==========================================================================
%%
function [out from to] = priv_getEnvChunk(from, to, signalENVB, useDelta1, ↩
    useDelta2, VERBOSE )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalENVB),
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalENVB)), '!' ]);
    end
    to = length(signalENVB);
end
if useDelta1 && useDelta2,
    out = signalENVB(from:to, :);
else
    if useDelta1,
        out = signalENVB(from:to, 1:16);
    else
        if useDelta2,
            out = signalENVB(from:to, [1:8 17:24]);
        else
            out = signalENVB(from:to, 1:8);
        end
    end
end
end
% ==========================================================================
%%
function [out from to] = priv_getMFCCChunk(from, to, signalMFCC, useDelta1, ↩
    useDelta2, VERBOSE )

if from < 1,
    if VERBOSE,
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣start␣index␣from␣', ...
            num2str(from), '␣to␣1!']);
    end
    from = 1;
end
if to > length(signalMFCC),
    if VERBOSE,
```

```
        warning('DD:INDEXOUTOFRANGE',['changing␣invalid␣end␣index␣from␣', ...
            num2str(to), '␣to␣' num2str(length(signalMFCC)), '!' ]);
    end
    to = length(signalMFCC);
end
if useDelta1 && useDelta2,
    out = signalMFCC(from:to, :);
else
    if useDelta1,
        out = signalMFCC(from:to, 1:16);
    else
        if useDelta2,
            out = signalMFCC(from:to, [1:8 17:24]);
        else
            out = signalMFCC(from:to, 1:8);
        end
    end
end
end
```

Listing B.65: functions/getTranscription.m

```
function [ newLabels ] = getTranscription( lexWORD, word )
% A little helper function to get the segment labels for the right context
% of the word's labelled target segments.
newLabels = cell(2,1);
switch lexWORD{word},

    case 'pranie'
        newLabels{1}='ni';
        newLabels{2}='e';

    case 'padnij'
        newLabels{1}='d';
        newLabels{2}='ni';

    case 'plamic'
        newLabels{1}='m';
        newLabels{2}='i';

    case 'labrys'
        newLabels{1}='b';
        newLabels{2}='r';

    case 'klawisz'
        newLabels{1}='w';
        newLabels{2}='i';

    case 'kadisz'
```

```
        newLabels{1}='d';
        newLabels{2}='i';

    case 'krasic'
        newLabels{1}='si';
        newLabels{2}='ci';

    case 'wikr'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'cypr'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'tir'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'typ'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'zupl'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'rabin'
        newLabels{1}='b';
        newLabels{2}='i';

    case 'tik'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'cykl'
        newLabels{1}='a';
        newLabels{2}='k';

    case 'gil'
        newLabels{1}='a';
        newLabels{2}='k';
end
end
```

Listing B.66: functions/cmatch.m

```
function [match left right] = cmatch( refContextLeft, candContextLeft, ...
    refContextRight, candContextRight )
```

```
3  %CMTACH context-match function of CSM
4  %   cm = cmatch( rcl, ccl, rcr, ccr)
5  %    returns the context-match value cm for the input arguments, reflecting
6  %    how similar the candidate context (i.e. the context in which the
7  %    exemplar was originally produced) is to the current production context
8  %    rcl = reference context left
9  %    ccl = candidate context left
10 %    rcr = reference context right
11 %    ccr = candidate context right
12 %   where, in its original implementation, left context is "acoustic" and
13 %   right context in "linguistic" (cf. Wade et al., 2010, J.Phon. 38).
14
15 %%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
16 %%% file created: 2010-11-08
17 %%%
18 %%% Based on original code by Travis Wade:
19 %%%   classes/@speaker/getSegCloud.m
20 %%%
21 %%% References:
22 %%% [wade&al] Wade, T.; Dogil, G.; Schütze, H.; Walsh, M. & Möbius, B.
23 %%%           "Syllable frequency effects in a context-sensitive segment
24 %%%           production model"; Journal of Phonetics, 2010, 38, 227-239
25 %%%
26
27 %%% - Original MATLAB code:
28 %%% thisContextMatch =
29 %%%  exp(sum(sum(prevContext.*candPrev))+sum(sum(follContext.*candFoll)));
30
31 %%% - Formula from [wade&al, p.235]:
32 %%% c_match(t_0; t_e; n_a; n_l; n_e) =
33 %%%  exp {
34 %%%   sum_{d=1}^{D_A}{ A_d, t_e - n_a : t_e - 1
35 %%%                    \cdot A_d, t_0 - n_a : t_0 - 1 }
36 %%%   +
37 %%%   sum_{d=1}^{D_L}{ L_d, t_e + n_e : t_e + n_e + n_l
38 %%%                    \cdot L_d, t_0 + n_e : t_0 + n_e + n_l }
39 %%%  }
40 %%%
41 %%% where L_d,m:n = (L_d,m; ...  L_d,n)^T (and analogously for A)
42
43 if nargout > 1,
44     left  = sum(sum( refContextLeft .* candContextLeft));
45     right = sum(sum(refContextRight .* candContextRight)) ;
46     match = exp( left + right);
47 else
48     match = exp( sum(sum( refContextLeft .* candContextLeft)) ...
49         + sum(sum(refContextRight .* candContextRight)) );
50 end
51 end
```

Listing B.67: functions/countBaselineStats.m

```
function [thisEvalBaselineSegment, thisEvalBaselineClass] = ...
    countBaselineStats( setup, segments, candx, targetx, ...
    thisEvalBaselineSegment, thisEvalBaselineClass)
%
% Updates:
%   [:,2] the total number of available candidates with matching left
%         context
%   [:,3] the total number of available candidates with matching right
%         context
%   [:,4] the total number of available candidates with matching left and
%         right contexts
%
% The total number of available candidates is determined outside based on
% the size of the candidate cloud.
%
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-09

% The label of the segment preceding the current target
% segment. This assumes that each utterance starts with
% a non-target segment, thus, t-1 is never 0.
tLeft = segments(targetx-1,3);

% The label of the segment following the current target
% segment. This assumes that each utterance starts with
% a non-target segment, thus, the last segment will
% always be followed bya SILENCE segment starting the
% next utterance. If the current target is the last
% segment in the corpus, we set the right context to
% SILENCE.
if targetx < size(segments,1),
    tRight = segments(targetx+1,3);
else
    tRight = setup.SIL;
end
if candx > 1,
    % The candidate segment selected for production is not the very first
    % segment in the corpus: get the label from thepreceding segment
    cLeft = segments(candx-1,3);
    % Check if the candidate segment is the very last segment in the
    % corpus:
    if candx < size(segments,1),
        cRight = segments(candx+1,3);
    else
        cRight = setup.SIL;
```

```
    end
else
    cLeft = setup.SIL;
    cRight = segments(candx+1,3);
end
% Geht the phone class
if cLeft > 0,
    cLeftClass  = setup.phoneToClass(cLeft);
else
    cLeftClass = setup.SILCLASS;
end
if cRight > 0,
    cRightClass = setup.phoneToClass(cRight);
else
    cRightClass = setup.SILCLASS;
end
if tLeft > 0,
    tLeftClass  = setup.phoneToClass(tLeft);
else
    tLeftClass = setup.SILCLASS;
end
if tRight > 0,
    tRightClass = setup.phoneToClass(tRight);
else
    tRightClass = setup.SILCLASS;
end
if cLeft == tLeft ,
    corrL = true;
    thisEvalBaselineSegment(2) = thisEvalBaselineSegment(2) + 1;
else
    corrL = false;
end
if cRight == tRight,
    corrR = true;
    thisEvalBaselineSegment(3) = thisEvalBaselineSegment(3) + 1;
else
    corrR = false;
end
if cLeftClass == tLeftClass,
    corrLc = true;
    thisEvalBaselineClass(2) = thisEvalBaselineClass(2) + 1;
else
    corrLc = false;
end
if cRightClass == tRightClass,
    corrRc = true;
    thisEvalBaselineClass(3) = thisEvalBaselineClass(3) + 1;
else
    corrRc = false;
```

```
end
if corrL && corrR,
    thisEvalBaselineSegment(4) = thisEvalBaselineSegment(4) + 1;
end
if corrLc && corrRc,
    thisEvalBaselineClass(4) = thisEvalBaselineClass(4) + 1;
end
end
```

B.3.2. Experiment with MOCHA corpus

This experiment shares most functions with the experiment with the Polish corpus. The following listings show only those parts which are implemented specifically for the experiment with the MOCHA corpus (cf. section A.4).

Listing B.68: runExperimentOnMocha.m

```
%%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
%%% file created: 2011-09-16

addpath('functions');
addpath('../io');
THISMFILENAME = 'runExperimentOnMocha';
% ==============================================================================
% Initialize setup
% ------------------------------------------------------------------------------
% create a struct to pass all global settings to any called
% function
setup = struct([]);
% save a time stamp for the debugging and logging output: this is
% the current date and time
setup(1).timestamp = datestr(now,'yyyy-mm-dd+HH-MM');
% ------------------------------------------------------------------------------
setup.FsA = 16000; % sampling frequency of audio signal files
% ------------------------------------------------------------------------------
setup.OUTDIR = '/mount/corpora12/a2/duran_working/diss/csm+ema/';
% INPUT %
setup.matlabDir = '/mount/corpora12/a2/duran_working/diss/db/mocha250/';
setup.speakerDirs = {'fsew0_v1.1/', 'msak0_v1.1/' };
% ------------------------------------------------------------------------------
% CSM context settings:
% context size (in seconds)
setup.leftContextSec  = 0.5;
setup.rightContextSec = 0;
% use non-normalized data or either length-normalized vectors or z-scores:
setup.lengthnormalize = true;
setup.znormalize      = false;
setup.useENVB = false;% use amplitude envelope bands
```

```
32 setup.useMFCC = true;
33 setup.useWAVE = false; % using the raw wave signal not supported yet(?)
34 % Delta1 and Delta2 are the signals' first and second derivatives:
35 setup.useDelta1 = true;% was: false
36 setup.useDelta2 = true;% was: false
37 % the data types to use:
38 % first column: articulatory data EMA
39 % second column: audio data ENV
40 % -> (1) EMA (2) ENV (3) EMA+ENV
41 setup.dataTypes = [ 1 0 ; 0 1 ; 1 1];
42 % print debugging info to console (and log file)
43 setup.VERBOSE=true;
44 setup.printProductionsToFile = false;
45 setup.EvalSylPos  = false;
46 setup.EvalSylType = false;
47 setup.EvalCA      = true; % context accuracy (phone labels and classes)
48 % ---------------------------------------------------------------------------
49 % The following parameters are used if context accuracy is evaluated, i.e.
50 % if setup.EvalCA==true
51 %
52 % Order of lexPHON sorted for printing the confusion matrices:
53 %     'SIL', 'breath',
54 %     'th', 'f', 's', 'sh', 'h', 'dh', 'v', 'z', 'zh',
55 %     'ch', 'jh',
56 %     'p', 't', 'k', 'b', 'd', 'g',
57 %     'm', 'n', 'ng',
58 %     'l', 'r', 'y', 'w',
59 %     'ii', 'iy', 'i', 'e', 'a',
60 %     '@', '@@', 'uh',
61 %     'aa', 'oo', 'o', 'u', 'uu',
62 %     'i@', 'ei', 'eir', 'ai', 'ow', 'oi', 'ou'
63 % (lexPHON is identical for speakers 1 and 2, therefore, defining only one
64 % sorting is ok for both)
65 setup.phoneSort = [ 1 2 ...
66     17 11 5 33 30 3 18 8 45 ...
67     44 22 ...
68     43 19 28 26 32 38 ...
69     25 24 29 ...
70     20 12 36 6 ...
71     9 10 4 35 40 ...
72     7 21 13 ...
73     31 14 34 42 23 ...
74     41 15 37 27 39 46 16 ]'; %'
75 % Broad phonetic classes for the categorisation of phones:
76 %  1 SL silence / other
77 %  2 FF voiceless fricatives
78 %  3 VV voiced fricatives
79 %  4 CH affricates
80 %  5 PP voicless plosives
```

```
81 %  6 BB voiced plosives
82 %  7 NM nasals
83 %  8 LR liquides
84 %  9 WY glides
85 % 10 VF vowels front
86 % 11 VM vowels mid
87 % 12 VB vowels back
88 % 13 DI diphthongs
89 setup.phoneClasses = {'SL'; 'FF'; 'VV'; 'CH'; 'PP'; 'BB'; 'NM'; 'LR'; 'WY'; 'VF';  ↩
      'VM';'VB'; 'DI' };
90 setup.phoneToClass = [1 1 3 10 2 9 11 3 10 10 2 8 11 12 13 13 2 3 5 8 11 4 12 7 7 6↩
       13 5 ... % sil...k
91      7 2 11 6 2 11 10 9 13 6 13 10 13 12 5 4 3 13]'; %'
92 setup.SILCLASS = 1;
93 % Labels of non-target segments which may be truncated in data extraction
94 % if the file durations do not match for audio and EMA data:
95 setup.nonTargetLex = {'sil'; 'breath'};
96 setup.SIL = 1;
97 % Function handle for the function which reads the data from the
98 % pre-processed corpus files and create the memory sequence data structures
99 % used in the CSM simulation:
100 setup.memorySequenceFH = @createMemorySequenceFromMocha ;
101 % ==============================================================================
102 % Check parameter consistency:
103 if ~ (setup.useENVB || setup.useMFCC),
104     error('Must␣select␣at␣least␣one␣audio␣representation␣(ENV␣or␣MFCC)!');
105 end
106 % ==============================================================================
107 % Settings for the synthesis of produced utterances:
108 setup.doSynthesis = false;
109 % if doSynthesis = false, the following settings have no effect:
110 setup.wavBase   = '/mount/corpora12/a2/duran/mocha/';
111 setup.wavSuffix = '.wav.snd';
112 % ==============================================================================
113 % generate less empty lines on output
114 format compact;
115 % set the number of threads for implicit multithreading to the
116 % maximum of available threads (beware angry colleagues)
117 maxNumCompThreads('automatic');
118 % set path and file name for the printing R script:
119 % (MATLAB cannot embedd fonts into PDF files)
120 RSCRIPT = '/home/users7/durandl/myCode/diss/emaClustering/R/printConfusionMatrix.R'↩
      ;
121 % ------------------------------------------------------------------------------
122 %% Working parameters (autosettings):
123 % Store indices of non-target segment labels for faster processing by
124 % avoiding string comparisons:
125 setup.nonTargetLabels = zeros(size(setup.nonTargetLex,1),1) ;
126 for sx=1:size(setup.nonTargetLex,1),
```

```
    setup.nonTargetLabels(sx)=sx;
end
setup.dataTypeLabels = cell(size(setup.dataTypes,1),1);
for dT=1:size(setup.dataTypes,1)
    lab = '';
    if setup.dataTypes(dT,1),
        lab = [lab '_EMA'];
    end
    if setup.dataTypes(dT,2),
        lab = [lab '_ENV'];
    end
    setup.dataTypeLabels{dT} = lab(2:end) ;
end
% ---------------------------------------------------------------------------
% set and create the base directory for this experiment's output:
setup.baseDir = [setup.OUTDIR, 'leftOnly/', setup.timestamp, '/' ] ;
diaryFilename = [setup.baseDir, THISMFILENAME, '.diary'];
% Create basic output directory for all parameter combindations:
mkdir(setup.baseDir);
% ---------------------------------------------------------------------------
% Start logging
diary(diaryFilename);
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']  Start: ', THISMFILENAME ]);
    disp(['  Saving diray to: ', diaryFilename]);
    disp( '  Setup: ');
    disp(setup);
    disp(['[', datestr(now,31), '] == START SIMULATION ==']);
end
csmProduction( setup );
disp(['[', datestr(now,31), '] All done. Start time was ', setup.timestamp, '. Bye!↩
    ']);
if setup.VERBOSE,
    disp(['[', datestr(now,31), '] Log file stored at: ', diaryFilename]);
end
diary off;
```

Listing B.69: functions/createMemorySequenceFromMocha.m

```
function [ corpus, labels, segments, files, rightContext, lex, targets ] = ...
    createMemorySequenceFromMocha( setup, speakerNr, useACC, useART )
%
% OUT: corpus     (N x D) whith N = number of frames,
%                   D = number of dimensions
%      labels     (N x 7) - used columns:
%                   [:,2] phone label index number
%                         or 0 for unlabeled parts of the corpus
%                   [:,6] file index
%                   [:,7] segment index
```

```
11 %      segments   optional:
12 %                 (S x 7), with columns:
13 %                   [:,1] start
14 %                   [:,2] end
15 %                   [:,3] label index number
16 %                   [:,4] file
17 %                   [:,5] syllable context = 0 (not used for MOCHA)
18 %                   [:,6] phone class
19 %                   [:,7] local start time in original file
20 %                   [:,8] local end time in original file
21 %      files      (F x 1) matrix
22 %      rightContext ...
23 %      lex          ...
24 %      targets    (S x 1) matrix
25
26 %%% Daniel Duran, SFB 732/A2, IMS Uni Stuttgart
27 %%% file created: 2011-09-16
28
29 %%% check input arguments:
30 rightContext=[];
31 lex={};
32 if nargin ~= 4,
33     error('Function␣must␣be␣called␣with␣exactly␣4␣input␣arguments!');
34 end
35 lengthnormalize = setup.lengthnormalize;
36 znormalize      = setup.znormalize;
37 if lengthnormalize && znormalize,
38     disp(['[', datestr(now,31), '␣', mfilename, ']␣' ...
39         'Cannot␣select␣both␣lengthnormalize␣and␣znormalize!␣'...
40         'Setting␣znormalize=false']);
41     znormalize = false;
42 end
43 getOriginalTime = setup.doSynthesis;
44 %%% check output arguments:
45 if nargout ~= 7,
46     error('␣createMemorySequenceFromMocha␣must␣be␣called␣with␣exactly␣7␣output␣↩
       arguments!');
47 end
48 doSeg = true;
49 if setup.rc > 0,
50     doRightContext = true;
51 else
52     doRightContext = false;
53 end
54 if setup.VERBOSE,
55     disp(['[', datestr(now,31), ',␣', mfilename, ']␣start']);
56     disp(['␣␣speakerNr:␣', num2str(speakerNr)]);
57     disp( '␣␣Setup:␣');
58     disp(setup);
```

```
59 end
60 %% start processing input directory -------------------------------------
61 % load index and label data for this speaker:
62 inDir = [setup.matlabDir setup.speakerDirs{speakerNr}];
63 load([inDir 'index.mat']);
64 %%% check whether the variables are loaded successfully
65 if ~exist('baseNames','var') || ~exist('checkFiles','var') || ...
66         ~exist('totalFramesEma','var') || ~exist('totalFramesWav','var')  || ...
67         ~exist('totalFramesEnv','var') || ~exist('totalFramesMFCC','var') || ...
68         ~exist('segmentsPHON','var')   ,
69     error('index.mat␣was␣not␣opened␣successfully!');
70 end
71 % ---------------------------------------------------------------------
72 offset=0;
73 accCols = [];
74 emaCols = [];
75 envCols = [];
76 mfcCols = [];
77 dTlog = '';
78 if useACC,
79     dTlog = 'ACC';
80     if setup.useWAVE,
81         disp('WARNING:␣adding␣WAVE␣not␣implemented!');
82     end
83     if setup.useMFCC,
84         plus = 13;
85         if setup.useDelta1 && setup.useDelta2,
86             plus=39;
87         else
88             if setup.useDelta1 || setup.useDelta2,
89                 plus=26;
90             end
91         end
92         mfcCols = (offset+1) : (offset+plus);% the MFCC columns in "corpus"
93         offset = offset + plus;
94     end
95     if setup.useENVB,
96         % use amplitude envelop representation (default for audio)
97         plus = 8;
98         if setup.useDelta1 && setup.useDelta2,
99             plus=24;
100        else
101            if setup.useDelta1 || setup.useDelta2,
102                plus=16;
103            end
104        end
105        envCols = (offset+1) : (offset+plus);% the envelop columns in "corpus"
106        offset = offset + plus;
107    end
```

```
    accCols = [mfcCols envCols];
end
if useART,
    dTlog = [dTlog 'ART'];
    plus = 8;
    if setup.useDelta1 && setup.useDelta2,
        plus=24;
    else
        if setup.useDelta1 || setup.useDelta2,
            plus=16;
        end
    end
    emaCols = (offset+1) : (offset+plus);
    offset = offset + plus;
end
% ---------------------------------------------------------------------------
% initialize output data
rows   = max([totalFramesEma, totalFramesEnv, totalFramesMFCC]) ;
corpus = zeros(rows, offset);
labels = zeros(rows,7);% columns: word, phon, tbo1, ttip, llip, file, segmentIndex
if doSeg,
    if getOriginalTime,
        % start \t end \t label \t file \t 0 \t startTime \t endTime
        segments = zeros(rows, 8);
    else
        % start \t end \t label \t file \t 0
        segments = zeros(rows, 6);
    end
    files = zeros( size(baseNames,1) , 1);
end
if setup.VERBOSE,
    disp(['[', datestr(now,31), '␣', mfilename, ']␣initialized␣coprus␣with␣'...
        num2str(rows), '␣rows␣and␣', num2str(offset), '␣columns' ]);
end
ptrAll    = 1;
ptrAllEnd = 1;
scnt = 0;% segment counter
fcnt = 0;% file counter
debugTooShortLC = 0;
% loop through utterances
for fdx=1:length(baseNames),
    bfile = baseNames{fdx};
    skip  = false;
    if setup.VERBOSE && mod(fdx,10)==0,
        disp(['[', datestr(now,31), '␣', mfilename, ']␣Get␣' dTlog ...
            '␣data␣from␣utterance␣#', num2str(fdx), '␣(', bfile, ')...']);
    end
    % the segments of this utterance:
    seg = segmentsPHON( segmentsPHON(:,4)==fdx , :);
```

```
157     % ---------------------------------------------------------------------
158     % Now, set the non-target labels to zero
159     %    targetSegments : (S x 1) matrix, where
160     %       S     = number of segments in current utterance
161     %       [:,1] = 1 if segment is target, else 0
162     targetSegments = zeros(size(seg,1),1);
163     % init pointers to first and last target segment in the
164     % current utterance #"fdx"
165     ptrFirstTargetSeg = size(seg,1)+1;
166     ptrLastTargetSeg  = 0;
167     for sidx =1:size(seg,1),
168         lab = seg(sidx,3);
169         if isempty(setup.nonTargetLabels(setup.nonTargetLabels==lab))
170             % The segment at position sidx is a target label
171             targetSegments(sidx)=1;
172             if sidx > ptrLastTargetSeg,
173                 ptrLastTargetSeg = sidx;
174             end
175             if sidx < ptrFirstTargetSeg,
176                 ptrFirstTargetSeg = sidx;
177             end
178         end %ENDIF
179     end %FOR
180     % ---------------------------------------------------------------------
181     ptrEmaEnd = 0;
182     ptrEnvEnd = 0;
183     labelsDone = false;
184     %%% Check whether all data is available
185     if useART,
186         if ~checkFiles(fdx,1),
187             if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣EMA␣↩
                   data']); end
188             skip = true ;
189         end
190     end
191     if useACC || setup.useWAVE || setup.useMFCC,
192         if ~checkFiles(fdx,2),
193             if setup.VERBOSE, disp(['Skipping␣file␣', baseNames{fdx}, '␣-␣no␣Audio␣↩
                   data']); end
194             skip = true ;
195         end
196     end
197     if skip,
198         continue;
199     end
200     % == EMA ======================================================== EMA ==
201     if useART,
202         % set pointer
203         ptrEma = ptrAll;
```

```
204         % load EMA data for the current utterance:
205         load([inDir bfile '_ema.mat']);
206         minEmaLength = min( [length(signalLLIPy), length(signalTBO1y), ...
207             length(signalTBO2y), length(signalTTIPy), ...
208             length(signalLLIPx), length(signalTBO1x), ...
209             length(signalTBO2x), length(signalTTIPx)] ) ;
210         load([inDir bfile '_frames.mat']);
211         % Go through the segments of the current utterance and
212         % add the EMA data to the output corpus
213         % Add unlabeled and non-target parts of the utterance if
214         % the left context size is >0
215         % Add unlabeled and non-target parts of the utterance if
216         % the right context size is >0
217         if setup.lc > 0 && ptrFirstTargetSeg > 1,
218             from = seg(ptrFirstTargetSeg,1) - setup.lc;
219             to   = seg(ptrFirstTargetSeg,1) - 1;
220             [from, to] = m_checkIndices(from, to, minEmaLength, baseNames{fdx}, ↩
                   setup.VERBOSE );
221             if size(signalLLIPx,1) < size(signalLLIPx,2),
222                 error(['signalLLIPx␣is␣row␣vector!␣File=' file]);
223             end
224             lc = priv_getChunk(from, to, setup.useDelta1, setup.useDelta2, ...
225                 signalLLIPy, signalTBO1y, signalTBO2y, signalTTIPy, ...
226                 signalLLIPx, signalTBO1x, signalTBO2x, signalTTIPx);
227             lcLength = size(lc,1);
228             zeropad = setup.lc - lcLength;
229             ptrZpad = 0;
230             if  zeropad > 0,
231                 % Not enough frames to form left context of the
232                 % specified size. Options:
233                 % (a) skip this utterance or (b) zero-padd
234
235                 debugTooShortLC = debugTooShortLC +1;
236
237                 % zero-padd at the beginning
238                 ptrEmaEnd = ptrEma + zeropad - 1;
239                 labels(ptrEma:ptrEmaEnd,6) = repmat( fdx, zeropad, 1);
240                 % The corresponding values in corpus and column
241                 % 1 of labels are already zero and need not to be
242                 % changed
243                 ptrZpad = ptrEma;
244                 ptrEma  = ptrEmaEnd + 1;
245             end
246             ptrEmaEnd = ptrEma + lcLength -1;
247             corpus(ptrEma:ptrEmaEnd, emaCols) = lc;
248             labels(ptrEma:ptrEmaEnd,6) = repmat( fdx, lcLength, 1);
249             labels(ptrEma:ptrEmaEnd,2) = frameLabels(from:to,:);
250             if doSeg,
251                 %  not adding unlabeled parts to the list of segments?
```

```
                scnt = scnt +1;
                if ptrZpad > 0,
                    segFrames = ptrZpad:ptrEmaEnd;
                else
                    segFrames = ptrEma:ptrEmaEnd;
                end
                lab = -1;
                if getOriginalTime,
                    segments(scnt,:) = [segFrames(1) segFrames(end) lab fdx 0 setup↩
                       .SILCLASS seg(1,5) seg(ptrFirstTargetSeg-1,6)];
                else
                    segments(scnt,:) = [segFrames(1) segFrames(end) lab fdx 0 setup↩
                       .SILCLASS ];
                end
                labels(ptrEma:ptrEmaEnd,7) = repmat( scnt, lcLength, 1);
                clear segFrames;
            end
            clear lc ptrEma ptrZpad zeropad;
        end % IF setup.lc
        % get segments:
        %context = 2; % 2=syllable onset
        for sidx = ptrFirstTargetSeg:size(seg,1),
            from = seg(sidx,1) ;
            to   = seg(sidx,2) ;
            [from, to] = m_checkIndices(from, to, minEmaLength, baseNames{fdx}, ↩
               setup.VERBOSE );
            sdat = priv_getChunk(from, to, setup.useDelta1, setup.useDelta2, ...
                signalLLIPy, signalTBO1y, signalTBO2y, signalTTIPy, ...
                signalLLIPx, signalTBO1x, signalTBO2x, signalTTIPx) ;
            ptrEma = ptrEmaEnd +1 ;
            ptrEmaEnd = ptrEma+size(sdat,1)-1;
            corpus(ptrEma:ptrEmaEnd, emaCols) = sdat;
            labels(ptrEma:ptrEmaEnd,6) = repmat( fdx, size(sdat,1), 1);
            labels(ptrEma:ptrEmaEnd,2) = frameLabels(from:to,:);
            if doSeg && sidx <= ptrLastTargetSeg,
                scnt = scnt +1;
                % Check whether the current segment is a non-target
                % (e.g. a silence label inside of an utterance)
                lab = seg(sidx,3);
                clz = setup.phoneToClass(lab);
                if isempty(setup.nonTargetLabels(setup.nonTargetLabels==lab))
                    % The current segment is a target
                else
                    % The current segment is not a target:
                    warning('DD:UnexpectedValue', ['Found␣non-target␣segment␣', ...
                        'inside␣utterance:␣', bfile, ';␣sidx=', num2str(sidx), ...
                        ';␣lab=', num2str(lab) ]);
                    lab=0;
                    clz = setup.SILCLASS;
```

```
                end
                if getOriginalTime,
                    segments(scnt, : ) = [ptrEma ptrEmaEnd lab fdx 0 clz seg(sidx↩
                       ,5) seg(sidx,6)];
                else
                    segments(scnt, : ) = [ptrEma ptrEmaEnd lab fdx 0 clz];
                end
                segFrames = ptrEma : ptrEmaEnd;
                labels(segFrames,7) = repmat(scnt, length(segFrames), 1);
            else
                % if sidx > ptrLastTargetSeg: --do nothing--
                % don't add right context (i.e. data after
                % the last target segment of the current
                % utterance to the list of segments!
            end
        end% FOR
        % get right context:
        if setup.rc>0,
            error('DD:NotImplemented', ...
                'Right␣context␣sizes␣>0␣not␣supported!');
        end
        labelsDone = true;
    end % useART
    % == ENV ======================================================== ENV ==
    if useACC,
        % set pointer
        ptrEnv = ptrAll;
        % load ENV data for the current utterance:
        file = [inDir bfile '_audio.mat'];
        load(file);
        framesFile = [inDir bfile '_frames.mat'];
        load(framesFile);
        minAccLength = min( length(signalMFCC), length(signalENVB) );
        if setup.lc > 0 && ptrFirstTargetSeg > 1,
            from = seg(ptrFirstTargetSeg,1) - setup.lc;
            to   = seg(ptrFirstTargetSeg,1) - 1;

            [from, to] = m_checkIndices(from, to, minAccLength, baseNames{fdx}, ↩
               setup.VERBOSE );
            if setup.useMFCC,
                lcM = priv_getMFCCChunk(from, to, signalMFCC, ...
                    setup.useDelta1, setup.useDelta2, setup.VERBOSE );
            else
                lcM=[];
            end
            if setup.useENVB,
                lcE = priv_getEnvChunk(from, to, signalENVB, ...
                    setup.useDelta1, setup.useDelta2,  setup.VERBOSE );
            else
```

```
                lcE=[];
            end
            lc = [lcM lcE];
            lcLength = size(lc,1);
            zeropad = setup.lc - lcLength;
            ptrZpad = 0;
            if  zeropad > 0,
                % Not enough frames to form left context of the
                % specified size. Options:
                % (a) skip this utterance or (b) zero-padd

                % zero-padd at the beginning
                ptrEnvEnd = ptrEnv + zeropad - 1;
                if ~labelsDone,
                    debugTooShortLC = debugTooShortLC +1;

                    labels(ptrEnv:ptrEnvEnd,6) = repmat( fdx, zeropad, 1);
                end
                ptrZpad = ptrEnv;
                ptrEnv  = ptrEnvEnd + 1;
            end
            ptrEnvEnd = ptrEnv+size(lc,1)-1;
            corpus(ptrEnv:ptrEnvEnd, accCols) = lc;
            if ~labelsDone,
                labels(ptrEnv:ptrEnvEnd,6) = repmat( fdx, lcLength, 1);
                labels(ptrEnv:ptrEnvEnd,2) = frameLabels(from:to,:);
                if doSeg,
                    %  not adding unlabeled parts to the list of
                    %  segments?
                    scnt = scnt +1;
                    if ptrZpad > 0,
                        segFrames = ptrZpad:ptrEnvEnd;
                    else
                        segFrames = ptrEnv:ptrEnvEnd;
                    end
                    lab = -1;
                    clz = setup.SILCLASS;
                    if getOriginalTime,
                        segments(scnt,:) = [segFrames(1) segFrames(end) lab fdx 0 ↩
                           clz seg(1,5) seg(ptrFirstTargetSeg-1,6)];
                    else
                        segments(scnt,:) = [segFrames(1) segFrames(end) lab fdx 0 ↩
                           clz];
                    end
                    labels(ptrEnv:ptrEnvEnd,7)   = repmat( scnt, lcLength, 1);
                    clear segFrames;
                end
            end
            clear lc ptrEnv ptrZpad zeropad;
```

```
392         end%IF
393         % get segments:
394         for sidx = ptrFirstTargetSeg:size(seg,1),%ptrLastTargetSeg,
395             from = seg(sidx,1) ;
396             to   = seg(sidx,2) ;
397             [from, to] = m_checkIndices(from, to, minAccLength, baseNames{fdx}, ↩
                   setup.VERBOSE );
398             if setup.useMFCC,
399                 sdatM = priv_getMFCCChunk(from, to, signalMFCC, ...
400                     setup.useDelta1, setup.useDelta2, setup.VERBOSE );
401             else
402                 sdatM=[];
403             end
404             if setup.useENVB,
405                 sdatE = priv_getEnvChunk(from, to, signalENVB, ...
406                     setup.useDelta1, setup.useDelta2,  setup.VERBOSE );
407             else
408                 sdatE=[];
409             end
410             sdat = [sdatM sdatE];
411             ptrEnv = ptrEnvEnd +1 ;
412             ptrEnvEnd = ptrEnv+size(sdat,1)-1;
413             corpus(ptrEnv:ptrEnvEnd, accCols) = sdat;
414             if ~labelsDone,
415                 labels(ptrEnv:ptrEnvEnd,6) = repmat( fdx, size(sdat,1), 1);
416                 labels(ptrEnv:ptrEnvEnd,2) = frameLabels(from:to,:);
417                 if doSeg && sidx <= ptrLastTargetSeg,
418                     scnt = scnt +1;
419                     % Check whether the current segment is a non-target
420                     % (e.g. a silence label inside of an utterance)
421                     lab = seg(sidx,3);
422                     clz = setup.phoneToClass(lab);
423                     if isempty(setup.nonTargetLabels(setup.nonTargetLabels==lab))
424                         % The current segment is a target
425                     else
426                         % The current segment is not a target:
427                         warning('DD:UnexpectedValue',['Found␣non-target␣segment␣', ↩
                               ...
428                             'inside␣utterance:␣', bfile ]);
429                         lab=0;
430                         clz = setup.SILCLASS;
431                     end
432                     if getOriginalTime,
433                         segments(scnt, : ) = [ptrEnv ptrEnvEnd lab fdx 0 clz seg(↩
                               sidx,5) seg(sidx,6)];
434                     else
435                         segments(scnt, : ) = [ptrEnv ptrEnvEnd lab fdx 0 clz];
436                     end
437                     segFrames = ptrEnv : ptrEnvEnd;
```

```
                    labels(segFrames,7) = repmat(scnt, length(segFrames), 1);
                else
                    % if sidx > ptrLastTargetSeg: --do nothing--
                    % don't add right context (i.e. data after
                    % the last target segment of the current
                    % utterance to the list of segments!

                    %end%IF <=ptrLastTargetSeg
                end%IF
            end %IF
        end %FOR sidx
        % get right context:
        if setup.rc>0,
            error('DD:NotImplemented', ...
                'Right␣context␣sizes␣>0␣not␣supported!');
        end
        labelsDone = true;
    end % useACC
    %%% current file ("utterance") done
    if doSeg,
        fcnt = fcnt +1;
        files(fcnt) = fdx;
    end
    ptrAllEnd = max([ptrEmaEnd ptrEnvEnd]);
    ptrAll = ptrAllEnd+1;
end% END FOR fdx
if setup.VERBOSE,
    disp(['[', datestr(now,31), ']␣<', mfilename, '>␣Found␣', ...
        num2str(debugTooShortLC), ...
        '␣utterances␣with␣too␣short␣left␣context']);
end
%%% trim output data
if ptrAllEnd < size(corpus,1),
    if setup.VERBOSE,
        disp(['[', datestr(now,31), '␣', mfilename, ']␣trimming␣corpus␣data␣from␣',↩
            num2str(size(corpus,1)), ...
            '␣rows␣down␣to␣', num2str(ptrAllEnd) ]);
    end
    corpus = corpus(1:ptrAllEnd,:);
    labels = labels(1:ptrAllEnd,:);
end
if doSeg,
    if scnt < size(segments,1),
        if setup.VERBOSE,
            disp(['[', datestr(now,31), '␣', mfilename, ']␣trimming␣segment␣data␣↩
               from␣', num2str(size(segments,1)), ...
                '␣rows␣down␣to␣', num2str(scnt) ]);
        end
        segments = segments(1:scnt,:);
```

```
    end
    if fcnt < size(files,1),
        if setup.VERBOSE,
            disp(['[', datestr(now,31), '␣', mfilename, ']␣trimming␣file␣index␣from↩
               ␣', num2str(size(files,1)), ...
                '␣rows␣down␣to␣', num2str(fcnt) ]);
        end
        files = files(1:fcnt,:);
    end
    %%% add right context labels
    if doRightContext,
        error('DD:NotImplemented', ...
            'Right␣context␣sizes␣>0␣not␣supported!');
    end % doRightContext
end
lex = lexPHON; %'
%%% normalize output data
if setup.lengthnormalize,
    if setup.VERBOSE, disp(['[', datestr(now,31), '␣', mfilename, ']␣length-↩
       normalizing␣data...']); end
    outDataNorm = zeros(size(corpus));
    for idx = 1:size(corpus,1) ,
        if useACC,
            if setup.useMFCC,
                if sum(corpus(idx,mfcCols)) ~= 0,
                    % Normalize only if the current vector is not 0
                    n = norm(corpus(idx,mfcCols));
                    nvec = corpus(idx,mfcCols) ./ n ;
                    outDataNorm(idx,mfcCols) = nvec;
                end
            end
            if setup.useENVB,
                % Warning: either check whether current vector is 0
                % E.g. if sum(corpus(idx,envCols)) ~= 0, ... end
                % or do not padd with 0 but with some low amplitude
                % gaussian noise!
                if sum(corpus(idx,envCols)) ~= 0,
                    % Normalize only if the current vector is not 0
                    n = norm(corpus(idx,envCols));
                    nvec = corpus(idx,envCols) ./ n ;
                    outDataNorm(idx,envCols) = nvec;
                end
            end
        end
        if useART,
            if sum(corpus(idx,emaCols)) ~= 0,
                n = norm(corpus(idx,emaCols));
                nvec = corpus(idx,emaCols) ./ n ;
                outDataNorm(idx,emaCols) = nvec;
```

```
            end
        end
        if sum(outDataNorm(idx,:)) ~=0,
            n = norm(outDataNorm(idx,:));
            nvec = outDataNorm(idx,:) ./ n ;
            outDataNorm(idx,:) = nvec;
        end
    end
    corpus =outDataNorm;
end
targets = zeros(size(segments,1),1);
lastUtt = 0;
for tidx = 1:size(segments,1),
    thisUtt = segments(tidx,4); % file
    if thisUtt ~= lastUtt,
        % this is the first segment of an utterance:
        % skip since we use it as left context
    else
        % this is not the first segment -> potential target
        lab = segments(tidx,3);
        if isempty(setup.nonTargetLabels(setup.nonTargetLabels==lab)),
            targets(tidx) = 1;
        end
    end
    lastUtt = thisUtt;
end
disp(['Total␣number␣of␣segments=' num2str(size(segments,1))]);
disp(['Number␣of␣targets=' num2str(sum(targets)) ]);
if setup.VERBOSE,
    disp(['[', datestr(now,31), '␣', mfilename, ']␣finished.']);
end
end

% ========================================================================
%% Function: check segment indices
function [from, to] = m_checkIndices(from, to, minSignalLength, fileName, VERBOSE )
if from < 1,
    if VERBOSE,
        disp(['␣␣Changing␣invalid␣start␣index␣from␣', num2str(from), ...
            '␣to␣1␣in␣file␣', fileName , '!']);
    end
    from = 1;
end
if to > minSignalLength,
    if VERBOSE,
        disp(['␣␣Changing␣invalid␣end␣index␣from␣', num2str(to), ...
            '␣to␣' num2str(minSignalLength), '␣in␣file␣', fileName, '!' ]);
    end
    to = minSignalLength;
```

```
end
end

% ==========================================================================
%% get chunk from the signal in the range [from:to]
function out = priv_getChunk(from, to, useDelta1, useDelta2, ...
    signalLLIPy, signalTBO1y, signalTBO2y, signalTTIPy, ...
    signalLLIPx, signalTBO1x, signalTBO2x, signalTTIPx)
off = 8;
if useDelta1 && useDelta2,
    dims = 24;
    off=16;
else
    if useDelta1 || useDelta2,
        dims = 16;
    else
        dims = 8;
    end
end
out = zeros(to - from +1, dims);
out(:,1) = signalLLIPx(from:to,1);
out(:,2) = signalLLIPy(from:to,1);
out(:,3) = signalTBO1x(from:to,1);
out(:,4) = signalTBO1y(from:to,1);
out(:,5) = signalTBO2x(from:to,1);
out(:,6) = signalTBO2y(from:to,1);
out(:,7) = signalTTIPx(from:to,1);
out(:,8) = signalTTIPy(from:to,1);
if useDelta1,
    out(:,9) = signalLLIPx(from:to,2);
    out(:,10) = signalLLIPy(from:to,2);
    out(:,11) = signalTBO1x(from:to,2);
    out(:,12) = signalTBO1y(from:to,2);
    out(:,13) = signalTBO2x(from:to,2);
    out(:,14) = signalTBO2y(from:to,2);
    out(:,15) = signalTTIPx(from:to,2);
    out(:,16) = signalTTIPy(from:to,2);
end
if useDelta2,
    out(:,off+1)  = signalLLIPx(from:to,3);
    out(:,off+2)  = signalLLIPy(from:to,3);
    out(:,off+3) = signalTBO1x(from:to,3);
    out(:,off+4) = signalTBO1y(from:to,3);
    out(:,off+5) = signalTBO2x(from:to,3);
    out(:,off+6) = signalTBO2y(from:to,3);
    out(:,off+7) = signalTTIPx(from:to,3);
    out(:,off+8) = signalTTIPy(from:to,3);
end
end
```

```
% ==========================================================================
function out = priv_getEnvChunk(from, to, signalENVB, useDelta1, useDelta2, debug )
if from < 1,
    if debug,
        disp(['WARNING: changing invalid start index from ', ...
            num2str(from), ' to 1!']);
    end
    from = 1;
end
if to > length(signalENVB),
    if debug,
        disp(['WARNING: changing invalid end index from ', ...
            num2str(to), ' to ' num2str(length(signalENVB)), '!' ]);
    end
    to = length(signalENVB);
end
if useDelta1 && useDelta2,
    out = signalENVB(from:to, :);
else
    if useDelta1,
        out = signalENVB(from:to, 1:16);
    else
        if useDelta2,
            out = signalENVB(from:to, [1:8 17:24]);
        else
            out = signalENVB(from:to, 1:8);
        end
    end
end
end

% ==========================================================================
function out = priv_getMFCCChunk(from, to, signalMFCC, useDelta1, useDelta2, ↩
    VERBOSE )
if useDelta1 && useDelta2,
    out = signalMFCC(from:to, :);
else
    if useDelta1,
        out = signalMFCC(from:to, 1:26);
    else
        if useDelta2,
            out = signalMFCC(from:to, [1:13 27:39]);
        else
            out = signalMFCC(from:to, 1:13);
        end
    end
end
end
```

Bibliography

Bas Aarts. Conceptions of gradience in the history of linguistics. *Language Sciences*, 26:343–389, 2004. (Referenced on page 167)

Guido Aversano, Anna Esposito, Antonietta Esposito, and Maria Marinaro. A new text-independent method for phoneme segmentation. In *Proceedings of the 44th IEEE 2001 Midwest Symposium on Circuits and Systems. MWSCAS 2001*, volume 2, pages 516–519, 2001. doi: 10.1109/MWSCAS.2001.986241. (Referenced on pages 66, 101 and 103)

Jolanta Bachan. Close copy speech synthesis for perception testing and annotation validation. Praca magisterska, Uniwersytet im. Adama Mickiewicza, Poznań, 2007. (Referenced on page 256)

Ladan Baghai-Ravary, Greg Kochanski, and John Coleman. Precision of phoneme boundaries derived using hidden markov models. In *Proceedings of the 10th Annual Conference of the International Speech Communication Association (Interspeech)*, pages 2879–2882, 2009. (Referenced on pages 83, 101 and 115)

Adam Baker. Computational approaches to the study of language change. *Language and Linguistics Compass*, 2(2):289–307, 2008. (Referenced on pages 36 and 59)

Martin Barbisch, Grzegorz Dogil, Bernd Möbius, Bettina Säuberlich, and Antje Schweitzer. Unit selection synthesis in the SmartWeb project. In *6th ISCA Workshop on Speech Synthesis (SSW6)*, pages 304–309, Bonn, Germany, 2007. URL http://www.isca-speech.org/archive_open/ssw6/ssw6_304.html. (Referenced on page 254)

Doug Beeferman, Adam Berger, and John Lafferty. Statistical models for text segmentation. *Machine Learning*, 34:177–210, 1999. (Referenced on page 105)

Robert C. Berwick, Paul Pietroski, Beracah Yankama, and Noam Chomsky. Poverty of the stimulus revisited. *Cognitive Science*, 35(7):1207–1242, 2011. (Referenced on page 85)

Hossein Bidgoli, editor. *Encyclopedia of Information Systems*. Academic Press, 2002. (Referenced on pages 435 and 444)

Steven Bird. Introduction to computational phonology. *Computational Linguistics*, 20(3):iii–ix, 1994. URL http://www.aclweb.org/anthology/J/J94/J94-3000. (Referenced on pages 49 and 50)

Peter Birkholz. *3D-Artikulatorische Sprachsynthese*. Logos Verlag Berlin, 2005. Dissertation, Universität Rostock. (Referenced on pages 57 and 62)

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, 2006. Corrected printing 2007. (Referenced on pages 65, 66, 68 and 165)

Louis-Jean Boë, Jean-Louis Heim, Kiyoshi Honda, Shinji Maeda, Pierre Badin, and Christian Abry. The vocal tract of newborn humans and neanderthals: Acoustic capabilities and consequences for the debate on the origin of language. A reply to Lieberman (2007a). *Journal of Phonetics*, 35:564–581, 2007. (Referenced on page 59)

Paul Boersma and Silke Hamann. The evolution of auditory dispersion in bidirectional constraint grammars. *Phonology*, 25:217–270, 2008. (Referenced on pages 36, 47, 48 and 59)

Michael R. Brent. Speech segmentation and word discovery: a computational perspective. *Trends in Cognitive Sciences*, 3(8):294–301, 1999a. (Referenced on pages 83, 87, 91 and 102)

Michael R. Brent. An efficient, probabilistically sound algorithm for segmentation and word discovery. *Machine Learning*, 34(1-3):71–105, Feb 1999b. doi: 10.1023/A:1007541817488. (Referenced on page 92)

Catherine P. Browman and Louis Goldstein. Some notes on syllable structure articulatory phonology. Status Report on Speech Research SR-93/94, Haskins Laboratories, 1988. URL http://www.haskins.yale.edu/sr/SR093/SR093_06.pdf. (Referenced on page 201)

Catherine P. Browman and Louis Goldstein. Articulatory phonology: An overview. Status Report on Speech Research SR-111/112, Haskins Laboratories, 1992. (Referenced on pages 50, 53 and 159)

Jagoda Bruni. *Sonorant voicing specification in phonetic, phonological and articulatory context*. Dissertation, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, 2011. (Referenced on pages 74, 158, 201, 222, 223 and 251)

Paul Cairns, Richard Shillcock, Nick Chater, and Joe Levy. Bootstrapping word boundaries: A bottom-up corpus-based approach to speech segmentation. *Cognitive Psychology*, 33:111–153, 1997. (Referenced on pages 66, 70, 91, 92 and 97)

Steve Cassidy. *The Emu Speech Database System*. Centre for Language Technology, Macquarie University, Sydney, Australia, 1.9 edition, 2004. URL http://emu.sourceforge.net/new_manual/index.html. Online manual. (Referenced on page 252)

Noam Chomsky. Language and nature. *Mind*, 104(413):1–61, Jan. 1995. New Series. (Referenced on page 247)

Michael H. Coen. Cross-modal clustering. In *Proceedings of the Twentieth AAAI Conference on Artificial Intelligence*, pages 932–937. American Association for Artificial Intelligence (AAAI), 2005. URL http://www.aaai.org/Library/AAAI/2005/aaai05-147.php. (Referenced on page 177)

Michael H. Coen. Self-supervised acquisition of vowels in american english. In *Proceedings of the Twenty-First AAAI Conference on Artificial Intelligence*, pages 1451–1456. Association for the Advancement of Artificial Intelligence (AAAI), 2006. URL http://www.aaai.org/Library/AAAI/2006/aaai06-228.php. (Referenced on pages 48 and 177)

Franklin S. Cooper. Speech synthesizers. In Sovijärvi and Aalto (1961), pages 3–13. (Referenced on pages 43, 51, 52 and 61)

Robert Daland and Janet B. Pierrehumbert. Learning diphone-based segmentation. *Cognitive Science*, 35 (1):119–155, 2011. doi: 10.1111/j.1551-6709.2010.01160.x. (Referenced on pages 66, 87, 88, 90, 91 and 99)

J. E. Dammann. An experiment in cluster detection [letter to the editor]. *IBM Journal of Research and Development*, 10(1):80–88, 1966. (Referenced on pages 176 and 178)

R. I. Damper. Self-learning and self-organization as tools for speech research. *Behavioral and Brain Sciences*, 21(2):262–263, 1998. Commentary / Sussman et al: Linear correlates in the speech signal. (Referenced on page 52)

E. E. David, Jr., Max V. Mathews, and H. S. McDonald. Digital computer simulation for speech experiments (abstract). *Journal of the Acoustical Society of America*, 31(1):113, 1959. URL http://dx.doi.org/10.1121/1.1930117. From: Program of the Fifty-Sixth Meeting of the Acoustical Society of America, November 20-22, 1958. (Referenced on pages 38, 39 and 43)

Bart de Boer. A realistic model of emergent phonology. AI-memo 98-04, Artificial Intelligence Laboratory at the Vrije Universiteit Brussel, 1998. (Referenced on pages 59 and 70)

Bart de Boer and W. Tecumseh Fitch. Computer models of vocal tract evolution: An overview and critique. *Adaptive Behavior*, 18:36–47, 2010. (Referenced on page 62)

Carl G. de Marcken. *Unsupervised Language Acquisition*. PhD thesis, Massachusetts Institute of Technology, September 1996. (Referenced on page 91)

Grażyna Demenko, Jolanta Bachan, Bernd Möbius, Katarzyna Klessa, M. Szymański, and S. Grocholewski. Development and evaluation of Polish speech corpus for unit selection speech synthesis systems. In *9th Annual Conference of the International Speech Communication Association (Interspeech)*. ISCA Archive, Spetember 2008a. URL http://www.isca-speech.org/archive/interspeech_2008/i08_1650.html. (Referenced on page 256)

Grażyna Demenko, Bernd Möbius, and Katarzyna Klessa. The design of Polish speech corpus for unit selection speech synthesis. In *Speech and Language Technology*, volume 11, pages 85–94. Polish Phonetic Association, 2008b. URL http://ptfon.pl/en/slt. (Referenced on page 256)

Ton Dijkstra and Koenraad de Smedt, editors. *Computational Psycholinguistics*. Taylor & Francis, 1996. (Referenced on pages 438 and 439)

Byron Dom. An information-theoretic external cluster-validity measure. In *Proceedings of the Eighteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 137–145, San Francisco, CA, 2002. Morgan Kaufmann. URL http://uai.sis.pitt.edu/papers/02/p137-dom.pdf. (Referenced on page 75)

Daniel Duran. Segmental foreign accent. Diplomarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, 2008. (Referenced on page 160)

Daniel Duran, Hinrich Schütze, and Michael Walsh. Evaluation measures for speech segmentation. Unpublished manusrcipt. (Referenced on pages 84 and 102)

Daniel Duran, Hinrich Schütze, and Bernd Möbius. Towards a computational model of unsupervised speech segmentation for correspondence learning. Poster, April 2010a. URL http://www.ifa.uni.wroc.pl/~glow33/PHONO-ABSTR.pdf. Presented at GLOW XXXIII (Positional phenomena in phonology and phonetics), Wrocław, Poland. (Referenced on page 84)

Daniel Duran, Hinrich Schütze, Bernd Möbius, and Michael Walsh. A computational model of unsupervised speech segmentation for correspondence learning. *Research on Language & Computation*, 8(2-3):133–168, 2010b. doi: 10.1007/s11168-011-9075-4. URL http://dx.doi.org/10.1007/s11168-011-9075-4. (Referenced on pages 56, 57, 66, 84, 139, 141, 149, 153, 154, 159, 163, 173, 194 and 239)

Daniel Duran, Jagoda Bruni, Grzegorz Dogil, and Hinrich Schütze. Speech events are recoverable from unlabeled articulatory data: Using an unsupervised clustering approach on data obtained from electromagnetic midsaggital articulography (EMA). In *Interspeech Conference Proceedings*, pages 2201–2204, Florence, Italy, August 2011a. (Referenced on pages 179, 181, 182 and 252)

Daniel Duran, Jagoda Bruni, Hinrich Schütze, and Grzegorz Dogil. Context sequence model of speech production enriched with articulatory features. In Lee and Zee (2011), pages 615–618. URL www.icphs2011.hk. (Referenced on pages 217, 238 and 252)

Daniel Duran, Jagoda Bruni, Hinrich Schütze, and Grzegorz Dogil. Using unlabeled EMA data in a speech production model with a rich memory. Presentation at: 7. Tagung zu Phonetik und Phonologie im deutschsprachigen Raum (P&P 7), October 2011c. URL http://www.home.uni-osnabrueck.de/tmeisenb/Duran_et_al.pdf. (Referenced on pages 217 and 252)

Daniel Duran, Jagoda Bruni, Hinrich Schütze, and Grzegorz Dogil. Specification in context – incorporation of an aritculatory factor into the context sequence model. In *43rd Poznań Linguistic Meeting (PLM 2012) – Book of Abstracts*, 2012a. URL http://ifa.amu.edu.pl/plm/2012/files/Abstracts/PLM2012_Abstract_Duran_etal.pdf. (Referenced on pages 217 and 252)

Daniel Duran, Jagoda Bruni, Michael Walsh, Hinrich Schütze, and Grzegorz Dogil. Phonological constraints verified by a rich memory exemplar model: extrametricality and articulatory binding in Polish obstruent-sonorant rhymes. In Antje Schweitzer and Britta Lintfert, editors, *Book of Abstracts of the 13th Conference on Laboratory Phonology, Stuttgart, Germany*, pages 217–218. Chair of Experimental Phonetics, Institute of Natural Language Processing, Universität Stuttgart, 2012b. Poster. (Referenced on pages 201 and 252)

Sorin Dusan and Lawrence Rabiner. On the relation between maximum spectral transition positions and phone boundaries. In *INTERSPEECH 2006 - ICSLP, Ninth International Conference on Spoken Language Processing*, pages 645–648, September 2006. URL http://www.isca-speech.org/archive/interspeech_2006/i06_1317.html. (Referenced on page 101)

David Ellis. A Bayesian model of natural language phonology: Generating alternations from underlying forms. In *Proceedings of the Tenth Meeting of ACL Special Interest Group on Computational Morphology and Phonology*, pages 12–19, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W08/W08-0703. (Referenced on page 67)

Attilio Erriquez, Rosa Caradonna, and Jacques Koreman. The application of a fuzzy clustering neural network to acoustic-phonetic mapping for ASR. *PHONUS*, 5:35–46, 2000. (Referenced on page 167)

Zsuzsanna Fagyal. Phonetics and speaking machines: On the mechanical simulation of human speech in the 17th century. *Historiographia Linguistica*, 28(3):289–330, 2001. (Referenced on page 42)

Gunnar Fant. Phonetics and phonology in the last 50 years. In Janet Slifka, Sharon Manuel, and Melanie Matthies, editors, *From Sound to Sense: 50+ Years of Discoveries in Speech Communication*, 2004. URL http://www.rle.mit.edu/soundtosense/conference/pdfs/invitedspeakers/Fant%20PAPER.pdf. (Referenced on pages 34, 43 and 52)

Naomi H. Feldman and Thomas L. Griffiths. A rational account of the perceptual magnet effect. In *Proceedings of the 29th Annual Conference of the Cognitive Science Society*, pages 257–262, 2007. (Referenced on pages 54 and 68)

Naomi H. Feldman, Thomas L. Griffiths, and James L. Morgan. The influence of categories on perception: Explaining the perceptual magnet effect as optimal statistical inference. *Psychological Review*, 116(4): 752–782, 2009a. (Referenced on pages 54, 68, 77 and 218)

Naomi H. Feldman, Thomas L. Griffiths, and James L. Morgan. Learning phonetic categories by learning a lexicon. In *Proceedings of the 31st Annual Conference of the Cognitive Science Society*, 2009b. (Referenced on page 68)

Margaret M. Fleck. Lexicalized phonotactic word segmentation. In *Proceedings of ACL-08: HLT*, pages 130–138, 2008. (Referenced on pages 87 and 89)

Carol A. Fowler. Speech as a supramodal or amodal phenomenon. In *The handbook of multisensory processes*, pages 189–201. The MIT Press, 2004. (Referenced on page 239)

Michael C. Frank, Sharon Goldwater, Vikash Mansinghka, Thomas L. Griffiths, and Joshua Tenenbaum. Modeling human performance in statistical word segmentation. In *Proceedings of the 29th Annual Meeting of the Cognitive Science Society*, pages 281–286, Nashville, Tennessee, USA, August 2007. URL http://csjarchive.cogsci.rpi.edu/proceedings/2007/docs/p281.pdf. (Referenced on page 88)

Paul L. Garvin. The automation of discovery procedure in linguistics. *Language*, 43(1):172–178, 1967. (Referenced on page 46)

Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, 1992. (Referenced on pages 153, 154 and 156)

Kevin Gold and Brian Scassellati. Audio speech segmentation without language-specific knowledge. In Ron Sun, editor, *The 28th Annual Conference of the Cognitive Science Society (CogSci 2006)*, pages 1370–1375, Vancouver, British Columbia, Canada, July 2006. Cognitive Science Society. URL http://csjarchive.cogsci.rpi.edu/proceedings/2006/docs/p1370.pdf. (Referenced on pages 97 and 98)

John Goldsmith and Aris Xanthos. Learning phonological categories. *Language*, 85(1):4–38, March 2009. doi: 10.1353/lan.0.0100. (Referenced on pages 48, 87, 167 and 176)

Sharon Goldwater, Thomas L. Griffiths, and Mark Johnson. A Bayesian framework for word segmentation: Exploring the effects of context. *Cognition*, 112(1):21–54, 2009. (Referenced on pages 67, 74, 87, 88, 91, 92, 93, 104 and 155)

Jan V. Goodsitt, James L. Morgan, and Patricia K. Kuhl. Perceptual strategies in prelingual speech segmentation. *Journal of Child Language*, 20:229–252, 1993. (Referenced on pages 88, 89 and 161)

Olga V. Goubanova. Bayesian modelling of vowel segment duration for text-to-speech synthesis using distinctive features. In *Proceedings of ICPhS 2003*, volume 3, page 2349, Barcelona, Spain, 2003. (Referenced on page 67)

Thomas L. Griffiths, Charles Kemp, and Joshua B. Tenenbaum. Bayesian models of cognition. In Sun (2008b). (Referenced on page 67)

Frank H. Guenther. Speech sound acquisition, coarticulation, and rate effects in a neural network model of speech production. *Psychological Review*, 1994a. (Referenced on page 62)

Frank H. Guenther. A neural network model of speech acquisition and motor equivalent speech production. *Biological Cybernetics*, 72:43–53, 1994b. (Referenced on pages 55 and 69)

Frank H. Guenther. Cortical interactions underlying the production of speech sounds. *Journal of Communication Disorders*, 39:350–365, 2006. doi: doi:10.1016/j.jcomdis.2006.06.013. (Referenced on pages 55 and 56)

Frank H. Guenther and Marin N. Gjaja. The perceptual magnet effect as an emergent property of neural map formation. *Journal of the Acoustical Society of America*, 100(2):1111–1121, 1996. (Referenced on pages 54, 55 and 69)

Frank H. Guenther and Tony Vladusich. A neural theory of speech acquisition and production. *Journal of Neurolinguistics*, 25(5):408–422, 2012. doi: doi:10.1016/j.jneuroling.2009.08.006. (Referenced on pages 55 and 56)

Frank H. Guenther, Alfonso Nieto-Castanon, Satrajit S. Ghosh, and Jason A. Tourville. Representation of sound categories in auditory cortical maps. *Journal of Speech, Language, and Hearing Research*, 47(1): 46–57, 2004. (Referenced on page 55)

Frank H. Guenther, Satrajit S. Ghosh, and Jason A. Tourville. Neural modeling and imaging of the cortical interactions underlying syllable production. *Brain and Language*, 96:208–301, 2006. (Referenced on pages 56 and 158)

Ibai Gurrutxaga, Javier Muguerza, Olatz Arbelaitz, Jesús M. Pérez, and José I. Martín. Towards a standard methodology to evaluate internal cluster validity indices. *Pattern Recognition Letters*, 32:505–515, 2011. doi: 10.1016/j.patrec.2010.11.006. (Referenced on pages 111, 169 and 170)

Edmund Gussmann. Resyllabification and delinking: The case of Polish voicing. *Linguistic Inquiry*, 23(1): 29–56, 1992. (Referenced on page 201)

Jacques B. M. Guy. Vowel identification: An old (but good) algorithm. *Cryptologie*, 15(3):258–262, 1991. (Referenced on page 48)

Sarah Hawkins. Roles and representations of systematic fine phonetic detail in speech understanding. *Journal of Phonetics*, 31(3-4):373–405, October 2003. (Referenced on pages 52, 86, 217 and 219)

William L. Henke. *Dynamic Articulatory Model of Speech Production Using Computer Simulation*. PhD thesis, Massachusetts Institute of Technology, 1966. (Referenced on pages 35, 46 and 62)

Ian S. Howard and Mark A. Huckvale. Training a vocal tract synthesizer to imitate speech using distal supervised learning. In *Speech and Computer (SPECOM '2005), Patras, Greece*, 2005. (Referenced on page 57)

Ian S. Howard and Piers Messum. Modeling the development of pronunciation in infant speech acquisition. *Motor Control*, 15:85–117, 2011. (Referenced on pages 57, 75, 158 and 177)

Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985. (Referenced on pages 172, 173 and 184)

William J. Idsardi. Poverty of the stimulus arguments in phonology. Unpublished manuscript, 2005. URL http://ling.umd.edu/~idsardi/papers/2005poverty.pdf. (Referenced on page 85)

Gerhard Jäger. Applications of game theory in linguistics. *Language and Linguistics Compass*, 2(3): 406–421, 2008. (Referenced on page 50)

Anil K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8):651–666, June 2010. doi: 10.1016/j.patrec.2009.09.011. (Referenced on pages 159, 160, 163, 165, 166 and 168)

Wiktor Jassem. Polish. *Journal of the International Phonetic Association*, 33(1):103–107, 2003. (Referenced on page 22)

Keith Johnson. Speech perception without speaker normalization: An exemplar model. In Keith Johnson and John Mullennix, editors, *Talker Variability in Speech Processing*, pages 145–165. Academic Press, 1997a. (Referenced on pages 52, 58, 76, 218, 219, 223 and 224)

Keith Johnson. The auditory/perceptual basis for speech segmentation. *OSU Working Papers in Linguistics*, 50:101–113, 1997b. URL http://linguistics.osu.edu/files/linguistics/workingpapers/osu_wpl_50.pdf. The Ohio State University – Department of Linguistics. (Referenced on pages 86, 95, 96, 97, 138, 154 and 247)

Keith Johnson. The linguistic basis of phonetic coherence. Annual report, UC Berkeley Phonology Lab, 2008. URL http://linguistics.berkeley.edu/phonlab/annual_report/documents/2008/Johnson_coherence.pdf. (Presented at the 11th Laboratory Phonology Conference, June 30, 2008). (Referenced on pages 52 and 86)

Peter W. Jusczyk. How infants begin to extract words from speech. *Trends in Cognitive Sciences*, 3(9): 323–328, Sept 1999. (Referenced on pages 88 and 92)

Samuel Kaski, Jari Kangas, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1981-1997. *Neural Computing Surveys*, 1:102–350, 1998. (Referenced on page 71)

Stuart A. Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, 1993. (Referenced on page 70)

Christopher T. Kello and David C. Plaut. A neural network model of the articulatory-acoustic forward mapping trained on recordings of articulatory parameters. *Journal of the Acoustical Society of America*, 116:2354–2364, 2004. (Referenced on pages 48, 52, 69, 158, 159, 213 and 247)

John L. Kelly, Jr. and Louis J. Gerstman. An artificial talker driven from a phonetic input (A). *Journal of the Acoustical Society of America*, 33(6):835, 1961. Abstract from: Program of the Sixty-First Meeting of the Acoustical Society of America; Session E. Speech Communications I. (Referenced on pages 44 and 61)

Thomas Kemp, Michael Schmidt, Martin Westphal, and Alex Waibel. Strategies for automatic segmentation of audio data. In *IEEE International Conference on Acoustics, Speech, and Signal Processing. ICASSP '00. Proceedings.*, volume 3, pages 1423–1426, 2000. doi: 10.1109/ICASSP.2000.861862. (Referenced on page 104)

Melody Y. Kiang. Neural networks. In Bidgoli (2002), pages 303–315. (Referenced on pages 66, 68, 69 and 71)

Simon King, Alan W. Black, Paul Taylor, Richard Caley, and Rob Clark. *Edinburgh Speech Tools Library: System Documentation Edition 1.2, for 1.2.3 24th Jan 2003*. Centre for Speech Technology, University of Edinburgh, 2003. URL http://www.cstr.ed.ac.uk/projects/speech_tools/manual-1.2.0/. (Referenced on page 257)

Robert Kirchner and Roger K. Moore. Computing phonological generalization over real speech exemplars. In *83rd Annual Meeting of the Linguistics Society of America*, 2009. (Referenced on page 176)

Dave Kleinschmidt and T. Florian Jaeger. A Bayesian belief updating model of phonetic recalibration and selective adaptation. Presented at the 17th Annual Conference on Architectures and Mechanisms for Language Processing (AMLaP), Paris, France, September 2011a. (Referenced on page 67)

Dave Kleinschmidt and T. Florian Jaeger. A Bayesian belief updating model of phonetic recalibration and selective adaptation. In *Proceedings of the 2nd Workshop on Cognitive Modeling and Computational Linguistics*, pages 10–19, Portland, Oregon, USA, June 2011b. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W11-0602. (Referenced on pages 49 and 67)

Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer Series in Information Sciences. Springer-Verlag, Berlin, Heidelberg, 3 edition, 1989. (Referenced on page 95)

Teuvo Kohonen. *Self-Organizing Maps*. Springer, Berlin, Heidelberg, New York, 3. edition, 2001. (Referenced on pages 36, 37, 68, 69, 70, 71, 72, 95 and 244)

Bernd J. Kröger. *Ein phonetisches Modell zur Sprachproduktion*. Max Niemeyer Verlag, Tübingen, 1998. (Referenced on page 62)

Bernd J. Kröger and Peter Birkholz. Articulatory synthesis of speech and singing: State of the art and suggestions for future research. In A. Esposito, A. Hussain, and M. Marinaro, editors, *Multimodal Signals: Cognitive and Algorithmic Issues*, Lecture Notes in Computer Science, pages 306–319. Springer, 2009. (Referenced on page 158)

Bernd J. Kröger, Peter Birkholz, and Christiane Neuschaefer-Rube. Ein neuronales Modell zur sensomotorischen Entwicklung des Sprechens. *Laryngo-Rhino-Otologie*, 86:365–370, 2007. doi: 10.1055/s-2006-944981. (Referenced on page 57)

Bernd J. Kröger, Jim Kannampuzhaa, and Christiane Neuschaefer-Rube. Towards a neurocomputational model of speech production and perception. *Speech Communication*, 51(9):793–809, 2009. doi: 10.1016/j.specom.2008.08.002. (Referenced on pages 57, 58, 69, 70, 72 and 73)

Bernd J. Kröger, Peter Birkholz, and Anja Lowit. Phonemic, sensory, and motor representations in an action-based neurocomputational model of speech production (ACT). In B. Maassen and P. van Lieshout, editors, *Speech Motor Control: New developments in basic and applied research*, chapter 2, pages 23–36. Oxford University Press, 2010. (Referenced on pages 58 and 158)

Henry Kučera. A note on the digital computer in linguistics. *Language*, 38(3):279–282, 1962. (Referenced on page 45)

Patricia K. Kuhl. Perception of speech and sound in early infancy. In Philip Salapatek and Leslie Cohen, editors, *Handbook of Infant Perception*, volume 2, pages 275–382. Academic Press, 1987. (Referenced on page 86)

Patricia K. Kuhl and Paul Iverson. Linguistic experience and the "Perceptual Magnet Effect". In Winifred Strange, editor, *Speech Perception and Linguistic Experience: Theoretical and Methodological Issues*. MD. York Press, 1995. (Referenced on pages 53 and 54)

Francisco Lacerda. The perceptual-magnet effect: An emergent consequence of exemplar-based phonetic memory. In K. Elenius and P. Branderyd, editors, *Proceedings of the 13th International Congress of Phonetic Sciences*, volume 2, pages 140–147, Stockholm, 1995. (Referenced on pages 54, 217 and 218)

Peter Ladefoged. Some possibilities in speech synthesis. *Language and Speech*, 7:205–214, 1964. (Referenced on pages 45, 46 and 62)

Sydney M. Lamb. The digital computer as an aid in linguistics. *Language*, 37(3):382–412, 1961. (Referenced on pages 33, 44, 45 and 59)

Wai-Sum Lee and Eric Zee, editors. *Proceedings of the ICPhS XVII*, August 2011. The organizers of ICPhS XVII at the Department of Chinese, Translation and Linguistics, City University of Hong Kong. URL `www.icphs2011.hk`. (Referenced on pages 432, 439 and 443)

Willem J. M. Levelt. Introduction. Hierarchical clustering algorithms in the psychology of grammar. In G. B. Flores d'Arcais and Willem J. M. Levelt, editors, *Advances in Psycholinguistics*, pages 101–108. North Holland, 1970. (Referenced on pages 165 and 168)

Natalie Lewandowski. *Talent in nonnative phonetic convergence*. Dissertation, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, 2012. URL `http://elib.uni-stuttgart.de/opus/volltexte/2012/7402/`. (Referenced on pages 38, 52 and 241)

Alvin M. Liberman and Ignatius G. Mattingly. The motor theory of speech perception revised. *Cognition*, 21:1–36, 1985. (Referenced on pages 53 and 159)

Johan Liljencrants and Björn Lindblom. Numerical simulation of vowel quality systems: The role of perceptual contrast. *Language*, 48(4):839–862, 1972. (Referenced on pages 26, 30, 46 and 47)

Ying Lin. Learning phonetic features from waveforms. Technical Report 103, Department of Linguistics, UCLA, 2004. URL `http://repositories.cdlib.org/uclaling/wpp/No103_5/`. (Referenced on pages 98 and 177)

Ying Lin. *Learning Features and Segments from Waveforms: A Statistical Model of Early Phonological Acquisition*. Dissertation, University of California, Los Angeles, 2005. URL `http://www.linguistics.ucla.edu/faciliti/research/YLin_diss.pdf`. (Referenced on pages 74, 98, 149 and 177)

Britta Lintfert, Antje Schweitzer, and Bernd Möbius. A parametric approach to intonation acquisition research: Validation on child-directed speech data. In *Interspeech Conference Proceedings*, pages 757–760, Florence, Italy, 2011. (Referenced on pages 176 and 177)

Leigh Lisker, Franklin S. Cooper, and Alvin M. Liberman. The uses of experiment in language description. *Word – Journal of the International Linguistic Association*, 18:82–106, 1962. (Referenced on page 62)

Andrej Ljolje, Julia Hirschberg, and Jan P. H. van Santen. Automatic speech segmentation for concatenative inventory selection. In Jan P. H. van Santen, Richard W. Sproat, Joseph P. Olive, and Julia Hirschberg, editors, *Progress in Speech Synthesis*, pages 305–312. Springer-Verlag, 1997. (Referenced on pages 83, 95, 101, 115 and 131)

B. M. Lobanov. Classification of Russian vowels spoken by different speakers. *Journal of the Acoustical Society of America*, 49(2):606–608, 1971. (Referenced on page 160)

Philipos C. Loizou, Michael Dorman, and Zhemin Tu. On the number of channels needed to understand speech. *Journal of the Acoustical Society of America*, 106(4):2097–2103, Oct 1999. (Referenced on page 181)

Shinji Maeda. Compensatory articulation during speech: Evidence from the analysis and synthesis of voal-tract shapes using an articulatory model. In William J. Hardcastle and Alain Marchal, editors, *Speech Production and Speech Modelling*, volume 55 of *NATO ASI Series D: Behavioural and Social Sciences*. Kluwer Academic Publishers, 1990. (Referenced on pages 57 and 62)

John Makhoul, Francis Kubala, Richard Schwartz, and Ralph Weischedel. Performance measures for information extraction. In *Proceedings of DARPA Broadcast News Workshop*, pages 249–252, 1999. (Referenced on page 104)

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press., Cambridge, MA, 1999. (Referenced on pages 37, 67, 68, 70, 77, 159 and 163)

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, online edition edition, 2009. URL `http://nlp.stanford.edu/IR-book/information-retrieval-book.html`. (Referenced on pages 22, 77, 163, 164, 165, 174, 175 and 184)

Dominic W. Massaro. Modelling multiple influences in speech perception. In Dijkstra and de Smedt (1996), chapter 4, pages 85–113. (Referenced on pages 51, 83, 85, 90 and 155)

Marina Meilă. Comparing clusterings—an information based distance. *Journal of Multivariate Analysis*, 98:873–895, 2007. (Referenced on pages 163, 168, 169 and 170)

Paul Mermelstein. Computer simulation of articulatory activity in speech production. In *Proceedings of the 1st international joint conference on Artificial intelligence (IJCAI'69)*, pages 447–454, 1969. (Referenced on pages 46 and 62)

Paul Mermelstein. Distance measures for speech recognition – psychological and instrumental. Status Report on Speech Research SR-47, Haskins Laboratories, 1976. URL `http://www.haskins.yale.edu/sr/sr047/SR047_07.pdf`. (Referenced on page 164)

N. Metropolis. The beginning of the Monte Carlo method. *Los Alamos Science*, 15:125–130, 1987. URL `http://library.lanl.gov/la-pubs/00326866.pdf`. Special Issue, Stanislaw Ulam 1909-1984. (Referenced on pages 44 and 74)

Werner Meyer-Eppler. *Elektrische Klangerzeugung*. Ferd. Dümmlers Verlag, Bonn, 1949. (Referenced on page 43)

Matthew Miller, Peter Wong, and Alexander Stoytchev. Unsupervised segmentation of audio speech using the voting experts algorithm. In *Proceedings of the 2nd Conference on Artificial General Intelligence (AGI-2009)*, Advances in Intelligent Systems Research. Atlantis Press, 2009. doi: 10.2991/agi.2009.25. (Referenced on pages 73, 75, 97, 98, 102 and 109)

Vikramjit Mitra, Hosung Nam, Carol Espy-Wilson, Elliot Saltzman, and Louis Goldstein. Recognizing articulatory gestures from speech for robust speech recognition. *Journal of the Acoustical Society of America*, 131(3):2270–2287, 2012. doi: 10.1121/1.3682038. (Referenced on pages 69, 73, 74, 153 and 158)

Kouki Miyazawa, Hideaki Miura, Hideaki Kikuchi, and Reiko Mazuka. The multi timescale phoneme acquisition model of the self-organizing based on the dynamic features. In *Interspeech Conference Proceedings*, pages 749–752, Florence, Italy, 2011. (Referenced on page 73)

Nelson Morgan, Hervé Bourlard, and Hynek Hermansky. Automatic speech recognition: An auditory perspective. In Steven Greenberg, William A. Ainsworth, Arthur N. Popper, and Richard R. Fay, editors, *Speech Processing in the Auditory System*, volume 18 of *Springer handbook of auditory research*, pages 309–338. Springer-Verlag, New York, 2004. (Referenced on pages 94 and 149)

Rebecca Morley. Bayesian learning over conflicting data: Predictions for language change. In *Proceedings of the Tenth Meeting of ACL Special Interest Group on Computational Morphology and Phonology (SIGMORPHON)*, pages 2–11, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/W/W08/W08-0702. (Referenced on page 67)

Doris Mücke, Jagoda Sieczkowska, Henrik Niemann, Martine Grice, and Grzegorz Dogil. Sonority profiles, gestural coordination and phonological licensing: Obstruent-sonorant clusters in Polish. Poster Presentation at the 12th Conference on Laboratory Phonology (LabPhon), Albuquerque, New Mexico, 2010. (Referenced on pages 201 and 251)

Jacob M. J. Murre and Rainer Goebel. Connectionist modelling. In Dijkstra and de Smedt (1996), pages 49–81. (Referenced on pages 66, 68 and 71)

Hosung Nam, Louis Goldstein, and Elliot Saltzman. Self organization of syllable structure: a coupled oscillator model. In François Pellegrino, Egidio Marsico, Ioana Chitoran, and Christophe Coupé, editors, *Approaches to phonological complexity*, pages 299–328. DE GRUYTER MOUTON, 2009. (Referenced on pages 50 and 51)

Daniel Nettle. Using social impact theory to simulate language change. *Lingua*, 108(2-3):95–117, 1999. (Referenced on page 60)

Dennis Norris and James M. McQueen. Shortlist B: A Bayesian model of continuous speech recognition. *Psychological Review*, 115(2):357–395, 2008. (Referenced on page 68)

John J. Ohala. Christian Gottlieb Kratzenstein: Pioneer in speech synthesis. In Lee and Zee (2011), pages 156–159. URL www.icphs2011.hk. (Referenced on page 42)

Merja Oja, Samuel Kaski, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 1998-2001 addendum. *Neural Computing Surveys*, 3:1–156, 2002. (Referenced on page 71)

M. Ostendorf. Moving beyond the 'beads-on-a-string' model of speech. In *Proc. IEEE ASRU Workshop*, pages 79–84, 1999. (Referenced on page 94)

Mari Ostendorf, Benoit Favre, Ralph Grishman, Dilek Hakkani-Tür, Mary Harper, Dustin Hillard, Julia Hirschberg, Heng Ji, Jeremy G. Kahn, Yang Liu, Sameer Maskey, Evgeny Matusov, Hermann Ney, Andrew Rosenberg, Elizabeth Shriberg, Wen Wang, and Chuck Wooters. Speech segmentation and spoken document processing. *IEEE Signal Processing Magazine*, 25(3):59–69, May 2008. (Referenced on page 93)

Pierre-Yves Oudeyer. *Self-Organization in the Evolution of Speech*. Number 6 in Studies in the Evolution of Language. Oxford University Press, Oxford, 2006. Translated by James R. Hurford. (Referenced on pages 59, 60 and 70)

Y. Pereiro Estevan, V. Wan, and Odette Scharenborg. Finding maximum margin segments in speech. In *ICASSP*, 2007. (Referenced on pages 103 and 168)

Amy Perfors, Joshua B. Tenenbaum, Thomas L. Griffiths, and Fei Xu. A tutorial introduction to Bayesian models of cognitive development. *Cognition*, 120:302–321, 2011. (Referenced on pages 67 and 69)

Franz Pernkopf, Tuan Van Pham, and Jeff A. Bilmes. Broad phonetic classification using discriminative Bayesian networks. *Speech Communication*, 51(2):151–166, 2009. (Referenced on page 67)

Pierre Perruchet and Ronald Peereman. The exploitation of distributional information in syllable processing. *Journal of Neurolinguistics*, 17:97–119, 2004. (Referenced on page 91)

Pierre Perruchet and Annie Vinter. Parser: A model for word segmentation. *Journal of Memory and Language*, 39:246–263, 1998. (Referenced on page 91)

Hagen Peukert. *Kindliche Kalkulationen*. Dissertation, Fachbereich Sprach- und Literaturwissenschaften der Universität Kassel, 2008. (Referenced on pages 85, 89, 90, 93 and 103)

Lev Pevzner and Marti A. Hearst. A critique and improvement of an evaluation metric for text segmentation. *Computational Linguistics*, 28(1):19–36, March 2002. URL http://www.aclweb.org/anthology/J/J02/J02-1002. (Referenced on page 105)

Janet B. Pierrehumbert. Exemplar dynamics: Word frequency, lenition, and contrast. In Joan L. Bybee and Paul Hopper, editors, *Frequency effects and the emergence of lexical structure*, pages 137–157. John Benjamins Publishing Company, 2001. (Referenced on pages 218, 219 and 240)

Max Planck. *Positivismus und reale Außenwelt*. Akademische Verlagsgesellschaft m. b. H., Leipzig, 1931. Vortrag gehalten am 12. November 1930 im Harnack-Haus der Kaiser Wilhelm-Gesellschaft zur Förderung der Wissenschaften. (Referenced on pages 36 and 83)

Matti Pöllä, Timo Honkela, and Teuvo Kohonen. Bibliography of self-organizing map (SOM) papers: 2002-2005 addendum. TKK Reports in Information and Computer Science TKK-ICS-R23, Helsinki University of Technology, Department of Information and Computer Science, 2009. URL http://lib.tkk.fi/Reports/2009/isbn9789522482532.pdf. (Referenced on page 71)

Robert F. Port and Adam P. Leary. Against formal phonology. *Language*, 81(4):927–964, 2005. (Referenced on pages 86 and 99)

Yu Qiao, Naoya Shimomura, and Nobuaki Minematsu. Unsupervised optimal phoneme segmentation: Objectives, algorithm and comparisons. In *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP*, pages 3989–3992, 2008. doi: 10.1109/ICASSP.2008.4518528. (Referenced on pages 101 and 175)

Okko Johannes Räsänen, Unto Laine Kalervo, and Toomas Altosaar. An improved speech segmentation quality measure: the R-value. In *Interspeech*, 2009. (Referenced on pages 102, 103 and 104)

Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 410–420, Prague, 2007. Association for Computational Linguistics. (Referenced on pages 164, 168, 170, 172, 174, 176 and 184)

Lawrence D. Rosenblum. Speech perception as a multimodal phenomenon. *Current Directions in Psychological Science*, 17(6):405–409, 2008. (Referenced on page 239)

Jenny R. Saffran, Richard N. Aslin, and Elissa L. Newport. Statistical learning by 8-month-old infants. *Science*, 274(5294):1926–1928, 1996. (Referenced on page 98)

Odette Scharenborg. Reaching over the gap: A review of efforts to link human and automatic speech recognition research. *Speech Communication*, 49:336–347, 2007. (Referenced on page 95)

Odette Scharenborg, Mirjam Ernestus, and Vincent Wan. Segmentation of speech: Child's play? In *Proceedings of Interspeech*, pages 1953–1956, Antwerp, 2007. (Referenced on page 168)

Odette Scharenborg, Vincent Wan, and Mirjam Ernestus. Unsupervised speech segmentation: An analysis of the hypothesized phone boundaries. *Journal of the Acoustical Society of America*, 127(2):1084–1095, 2010. (Referenced on pages 98, 99, 101, 102, 103, 131 and 176)

M. R. Schroeder. Computers in acoustics: Symbiosis of an old science and a new tool. *Journal of the Acoustical Society of America*, 45(5):1077–1088, 1969. (Referenced on page 43)

Antje Schweitzer. Clusterexperimente zur Identifikation von prosodischen Kategorien. Presentation at P&P 6 (6. Tagung zu Phonetik und Phonologie im deutschsprachigen Raum), October 2010a. URL http://web.phonetik.uni-frankfurt.de/pundp/PundP_Abstracts/Abstract_Schweitzer.pdf. (Referenced on page 77)

Antje Schweitzer. *Production and Perception of Prosodic Events – Evidence from Corpus-based Experiments*. Dissertation, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, 2010b. (Referenced on pages 176 and 254)

Antje Schweitzer, Norbert Braunschweiler, Grzegorz Dogil, Tanja Klankert, Bernd Möbius, Gregor Möhler, Edmilson Morais, Bettina Säuberlich, and Matthias Thomae. Multimodal speech synthesis. In Wolfgang Wahlster, editor, *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 411–435. Springer, 2006. (Referenced on pages 148 and 254)

C. E. Shannon. A mathematical theory of communication [part 1]. *The Bell System Technical Journal*, 27(3):379–423, July 1948. URL http://www.alcatel-lucent.com/bstj/vol27-1948/articles/bstj27-3-379.pdf. (Referenced on page 78)

Manish Sharma and Richard J. Mammone. Subword-based text-dependent speaker verification system with user-selectable passwords. In *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP-96). Conference Proceedings.*, pages 93–96, May 1996a. doi: 10.1109/ICASSP.1996.540298. (Referenced on pages 83, 97, 98 and 109)

Manish Sharma and Richard J. Mammone. "Blind" speech segmentation: Automatic segmentation of speech without linguistic knowledge. In *Fourth International Conference on Spoken Language, 1996. ICSLP 96. Proceedings.*, volume 2, pages 1237–1240, Philadelphia, PA , USA, 1996b. doi: 10.1109/ICSLP.1996.607832. (Referenced on pages 66, 83, 97 and 109)

Lei Shi, Thomas L. Griffiths, Naomi H. Feldman, and Adam N. Sanborn. Exemplar models as a mechanism for performing Bayesian inference. *Psychonomic Bulletin & Review*, 17(4):443–464, 2010. (Referenced on pages 55, 68, 74 and 218)

Jorge Silva, Vivek Rangarajan, Viktor Rozgić, and Shrikanthtop Narayanan. Information theoretic analysis of direct articulatory measurements for phonetic discrimination. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume IV, pages 457–460, April 2007. doi: 10.1109/ICASSP.2007.366948. (Referenced on page 158)

Malcolm Slaney. Auditory toolbox. Software library. URL `http://cobweb.ecn.purdue.edu/~malcolm/interval/1998-010/`. Accessed online: 2009-06. (Referenced on pages 148 and 223)

A. Solomonoff, A. Mielke, M. Schmidt, and H. Gish. Clustering speakers by their voices. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, 1998.*, volume 2, pages 757 – 760, Seattle, WA, 1998. doi: 10.1109/ICASSP.1998.675375. (Referenced on pages 174 and 175)

Antti Sovijärvi and Pentti Aalto, editors. *Proceedings of the Fourth International Congress of Phonetic Sciences*, September 1961. Mouton & Co. (Referenced on pages 43 and 430)

Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. Technical Report 00-034, University of Minnesota - Computer Science and Engineering, 2000a. URL `http://www.cs.umn.edu/research/technical_reports.php?page=report&report_id=00-034`. (Referenced on pages 154, 165, 168, 169, 171, 172 and 184)

Michael Steinbach, George Karypis, and Vipin Kumar. A comparison of document clustering techniques. In *KDD Workshop on Text Mining*, 2000b. Poster. (Referenced on pages 183 and 184)

Veronique Stouten, Kris Demuynck, and Hugo Van hamme. Discovering phone patterns in spoken utterances by non-negative matrix factorization. *Signal Processing Letters, IEEE*, 15:131–134, 2008. (Referenced on page 96)

Alexander Strehl. *Relationship-based Clustering and Cluster Ensembles for High-dimensional Data Mining*. Dissertation, University of Texas, Austin, May 2002. URL `http://www.lans.ece.utexas.edu/~strehl/diss/`. (Referenced on page 174)

Ron Sun. Introduction to computational cognitive modeling. In *The Cambridge Handbook of Computational Psychology* Sun (2008b), chapter 1, pages 3–19. (Referenced on page 35)

Ron Sun, editor. *The Cambridge Handbook of Computational Psychology*. Cambridge University Press, New York, NY, USA, 2008b. (Referenced on pages 433 and 442)

Daniel Swingley. Statistical clustering and the contents of the infant vocabulary. *Cognitive Psychology*, 50: 86–132, 2005. (Referenced on pages 87, 88, 90 and 97)

the International Phonetic Association, editor. *The Handbook of the International Phonetic Association: A Guide to the Use of the International Phonetic Alphabet*. Cambridge University Press, 1999. (Referenced on page 22)

The MathWorks, Inc. Signal processing toolbox, version 6.11 (r2009a). Software package for MATLAB, 2009. (Referenced on page 234)

The R Foundation for Statistical Computing. R, version 2.14. Software environment, 2011. URL http://www.r-project.org/. (Referenced on page 160)

Margaret Thomas. Development of the concept of 'the poverty of the stimulus'. *The Linguistic Review*, 19 (1-2):51–71, January 2002. ISSN 0167-6318, 1613-3676. (Referenced on page 85)

Sam Tilsen. Metrical regularity facilitates speech planning and production. *Laboratory Phonology*, 2(1): 185–218, 2011. (Referenced on pages 51 and 78)

Martine Toda and Shinji Maeda. Quantal aspects of non anterior sibilant fricatives: a simulation study. In Hani Camille Yehia, Didier Demolin, and Rafael Laboissière, editors, *Proceedings of the 7th International Seminar on Speech Production (ISSP '06)*, Ubatuba, SP, Brazil, December 2006. (Referenced on page 62)

Doroteo Torre Toledano, Luis A. Hernández Gómez, and Luis Villarrubia Grande. Automatic phonetic segmentation. In *IEEE Transactions on Speech and Audio Processing*, volume 11, pages 617–625, 2003. (Referenced on pages 83 and 101)

F. Trautwein. Über elektrische Synthese von Sprachlauten und musikalischen Tönen. In unknwon ICPhS editor, editor, *Proceedings of the International Congress of Phonetic Sciences*, pages 200–201, Amsterdam, July 1932. (Referenced on page 43)

Jürgen Trouvain and Fabian Brackhane. Wolfgang von Kempelen's 'speaking machine' as an instrument for demonstration and research. In Lee and Zee (2011), pages 164–167. URL www.icphs2011.hk. (Referenced on page 42)

Alan M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, Oct. 1950. New Series. (Referenced on pages 44 and 248)

C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979. Second Edition. (Referenced on page 77)

Maarten van Segbroeck and Hugo Van hamme. Unsupervised learning of time–frequency patches as a noise-robust representation of speech. *Speech Communication*, 51:1124–1138, 2009. (Referenced on pages 49 and 97)

Balakrishnan Varadarajan, Sanjeev Khudanpu, and Emmanuel Dupoux. Unsupervised learning of acoustic sub-word units. In *Proceedings of ACL-08: HLT, Short Papers (Companion Volume)*, pages 165–168, Columbus, Ohio, June 2008. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/P/P08/P08-2042. (Referenced on page 96)

Travis Wade and Bernd Möbius. Speaking rate effects in a landmark-based phonetic exemplar model. In *8th Annual Conference of the International Speech Communication Association — Interspeech*, pages 402–405, Aug 2007. URL http://www.isca-speech.org/archive/interspeech_2007/i07_0402.html. (Referenced on pages 220, 223 and 240)

Travis Wade, Grzegorz Dogil, Hinrich Schütze, Michael Walsh, and Bernd Möbius. Syllable frequency effects in a context-sensitive segment production model. *Journal of Phonetics*, 38(2):227–239, 2010. (Referenced on pages 98, 137, 138, 141, 150, 153, 181, 217, 220, 221, 222, 223, 237, 239 and 240)

Michael Walsh, Bernd Möbius, Travis Wade, and Hinrich Schütze. Multilevel exemplar theory. *Cognitive Science*, 34:537–582, 2010. (Referenced on pages 57 and 241)

Andrew B. Wedel. Category competition drives contrast maintenance within an exemplar-based production/perception loop. In *Proceedings of the Seventh Meeting of the ACL Special Interest Group in Computational Phonology*, pages 1–10, Barcelona, Spain, July 2004a. Association for Computational Linguistics. (Referenced on pages 47, 53 and 219)

Andrew B. Wedel. *Self-Organization and Categorical Behavior in Phonology*. Dissertation, University of California, Santa Cruz, March 2004b. (Referenced on pages 36, 47 and 75)

Andrew B. Wedel. Feedback and regularity in the lexicon. *Phonology*, 24:147–185, 2007. (Referenced on page 53)

Andrew B. Wedel. Variation, multi-level selection and conflicts between phonological and morphological regularities. In Juliette Blevins, editor, *Analogy in Grammar: Form and Acquisition*. Oxford University Press, 2009. (Referenced on pages 59 and 155)

Andrew B. Wedel. Self-organization in phonology. In Marc van Oostendorp, Colin J. Ewen, Elizabeth Hume, and Keren Rice, editors, *The Blackwell Companion to Phonology*, chapter 6, pages 130–147. Blackwell Publishing, 2011. (Referenced on page 70)

Maria-Barbara Wesenick and Andreas Kipp. Estimating the quality of phonetic transcriptions and segmentations of speech signals. In *Fourth International Conference on Spoken Language, 1996. ICSLP 96. Proceedings.*, volume 1, pages 129–132, Philadelphia, PA , USA, 1996. doi: 10.1109/ICSLP.1996.607054. (Referenced on pages 101, 134 and 156)

Ludwig Wittgenstein. *Philosophische Untersuchungen*. Suhrkamp Taschenbuch Verlag, 1971. (Referenced on page 167)

Alan A. Wrench. MOCHA-TIMIT – MultiCHannel Articulatory database: English. Speech database, available online, November 1999. URL `http://www.cstr.ed.ac.uk/research/projects/artic/mocha.html`. Accessed 2011-05-26. (Referenced on pages 234 and 257)

Alan A. Wrench. A multi-channel/multi-speaker articulatory database for continuous speech recognition research. In W.J. Barry, J. Koreman, and K. Kirchhof, editors, *Phonus*, volume 5, pages 1–13. Institute of Phonetics, Saarland University, Saarbrücken, 2000. (Referenced on pages 158 and 257)

Aris Xanthos. An incremental implementation of the utterance-boundary approach to speech segmentation. In *Proceedings of the 14th Meeting of Computational Linguistics in the Netherlands*, pages 171–180, 2003. URL `http://www.cnts.ua.ac.be/clin2003/`. (Referenced on pages 87, 88, 89 and 91)

Linli Xu, James Neufeld, Bryce Larson, and Dale Schuurmans. Maximum margin clustering. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1537–1544, Cambridge, MA, 2005. MIT Press. URL `http://books.nips.cc/papers/files/nips17/NIPS2004_0834.pdf`. NIPS*2004. (Referenced on pages 176 and 177)

K. Yoshida and A. Sakurai. Machine learning. In Bidgoli (2002), pages 103–114. (Referenced on page 176)

Bartosz Ziółko, Suresh Manandhar, and Richard Wilson. Fuzzy recall and precision for speech segmentation evaluation. In *Proceedings of 3rd Language & Technology Conference, Poznań, Poland*, 2007. (Referenced on page 106)

Willem Zuidema and Bart de Boer. The evolution of combinatorial phonology. *Journal of Phonetics*, 37(2): 125–144, 2009. (Referenced on page 50)

Author Index

This is a list of *all* authors whose work is referenced in the text, sorted by *last name*, *first name*, with the corresponding page numbers. This list does not distinguish between first and other authors.

Erklärung

Hiermit erkläre ich, dass ich außer den ausdrücklich bezeichneten Hilfsmitteln die Dissertation selbständig verfasst habe.

Daniel Duran