Kompressionsbasierte Mustererkennung

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

Vorgelegt von

Sebastian Klenk

aus Stuttgart

Hauptberichter:	Prof. Dr. G. Heidemann
Mitberichter:	Prof. Dr. T. Ertl
Tag der mündlichen Prüfung:	10.12.2012

Institut für Visualisierung und Interaktive Systeme
der Universität Stuttgart

2012

Wir müssen wissen.
Wir werden wissen.

David Hilbert

VORWORT

Kompressionsbasierte Mustererkennung ist ein Bereich der Mustererkennung, der bisher nur von einer Seite – der der Kolmogorovkomplexität – beleuchtet wurde. In dieser Arbeit soll, durch theoretische und experimentelle Überlegungen, ein neuer Blickwinkel auf dieses Fachgebiet ermöglicht werden. Diese Bestrebung findet sich auch in der Struktur der Arbeit wieder. Im Folgenden werden wir einen kurzen Leitfaden durch die Kapitel dieser Arbeit legen, sodass wir schnell zu den jeweils interessanten Abschnitten kommen. Darüber hinaus soll dieser Leitfaden auch als Wegweiser zu den eigenen Beiträgen des Autors dienen.

Kapitel 1, **Einleitung** gibt einen Überblick über das gesamte behandelte Themenspektrum. Dort erhalten wir einen leichten Einstig in das Gebiet der kompressionsbasierten Mustererkennung.

Kapitel 2 (Normalized Compression Distance), **Theorie der NCD** beschäftigt sich in erster Linie mit der Kolmogorovkomplexität und deren Implikationen. Dort findet neben einer kurzen Erklärung der Konzepte der Kolmogorovkomplexität auch eine kritische Auseinandersetzung mit dem Thema statt. Das Korollar 2.1 und der Satz 2.3 sowie die Beispiele 2.2 und 2.3 hinterfragen zum ersten Mal auch die theoretische Rechtfertigung der NCD.

Eigener Beitrag: Dieses Kapitel dient, bis auf die gerade erwähnten Beiträge, hauptsächlich der Auseinadersetzung mit dem Stand der Technik und zitiert im wesentlichen existierende Arbeiten anderer Wissenschaftler.

Kapitel 3, **Praxis der NCD** beleuchtet die praktischen Aspekte der NCD. Hier finden Sie neben einer ausführlichen Erläuterung zu den einzelnen Kompressionsverfahren auch eine Beispielanwendung der NCD. Besonders interessant ist hier die Anwendung in Abschnitt 3.2.1 und die zahlreichen Experimente in Abschnitt 3.2.2.

Eigener Beitrag: Neben dem Anwendungsszenario sind besonders die durchgeführten Experimente hervorzuheben, die, in dieser Form einzigartig, die Funktionsweise der NCD erforschen. Die Ergebnisse dieser Experimente bilden die Basis für die Überlegungen der nachfolgenden Kapitel.

Kapitel 4, **Theorie der kompressionsbasierten Mustererkennung** legt das Fundament für die spätere praktische Anwendung der kompressionsbasierten Mustererkennung. Dabei liefert die Behandlung von Vektorräumen über redundanten Wörterbüchern (Abschnitt 4.2) eine fundamental neue Sichtweise auf das Gebiet des *Sparse Codings*. Die Ausführungen zur kompressionsbasierten Mustererkennung (Abschnitt 4.6) bauen ebenfalls auf diesen Erkenntnissen auf und erlauben erstmalig eine allumfassende Theorie zur kompressionsbasierten Mustererkennung. Darüber hinaus findet sich in diesem Kapitel auch die Grundlage für die allgemeine Behandlung nichtvektorieller Daten in Vektorräumen.

Eigener Beitrag: Die in diesem Kapitel angestellten Überlegungen, dünne Repräsentationen als Elemente einer eigenständigen Vektorraumstruktur zu betrachten, sind nach bestem Wissen des Autors einzigartig. Die aus diesen Überlegungen entwickelten Verfahren zur kompressionsbasierten Mustererkennung stellen ebenfalls eine neue Herangehensweise dar. Ebenso verhält es sich mit den Verfahren zur allgemeinen Behandlung nichtvektorieller Daten in Vektorräumen.

Kapitel 5, **Praxis der kompressionsbasierten Mustererkennung**, untermauert die theoretischen Erkenntnisse der vorangegangenen Kapitel. Dabei stehen zwei Fragestellungen im Vordergrund:

1. Kann die Theorie der kompressionsbasierten Mustererkennung die Erfolge der NCD beschreiben und reproduzieren und

2. wie geeignet ist diese Theorie für die Anwendung auf kontinuierliche Daten.

Eigener Beitrag: Die Experimente zur kompressionsbasierten Mustererkennung basieren alle auf den Überlegungen der vorangegangenen Kapitel.

Kapitel 6, **Zusammenfassung**, gibt nochmal einen Überblick über die gesamte Arbeit und fasst alle Ergebnisse zusammen.

Neben diesem Leitfaden ist jedem Kapitel auch noch eine kurze Einführungsbox vorangestellt. In dieser werden die wesentlichen Punkte des Kapitels aufgeführt, und es gibt Referenzen zu den wichtigsten Abschnitten.

Überblick
Eine Übersichtsbox wie diese gibt eine kurze Einführung zu dem nachfolgenden Kapitel und weist auf wichtige Aspekte hin.

Außer dem hier vorgestellten Thema der Kompressionsbasierten Mustererkennung habe ich mich wärend meiner Dissertation auch noch mit der Visualisierung von Merkmalen (Shoukry et al. 2010, Heidemann und Klenk 2007, Imo et al. 2008a und Imo et al. 2008b) und der Analyse medizinischer Daten beschäftigt (Fritz et al. 2010, Klenk et al. 2009a, Klenk et al. 2010a, Klenk et al. 2010b, Klenk et al. 2011b, Klenk et al. 2011a, Klenk et al. 2011c, Klenk et al. 2008 und Fritz et al. 2008).

Diese Arbeit wäre ohne die tatkräftige Unterstützung zahlreicher Personen nicht möglich gewesen. Danken möchte ich daher meinen Kollegen Andre Burkovski und Dennis Thom, die durch ihre eigene Arbeit und regelmäßige angeregte Diskussionen einen großen Beitrag zu den hier aufgeführten Ideen und Experimenten geliefert haben.

Besonders danken möchte ich auch meinem Doktorvater Prof. Gunther Heidemann, der mich immer unterstützt hat eigene Ideen zu verfolgen und diese mit seiner Erfahrung und guten Ratschlägen angereichert hat, sowie, PD Dr. Jürgen Dippon

für sein scharfes Auge, seine Geduld und, genauso wie Dr. Peter Fritz, für seinen unerschütterlichen Glauben an Wissenschaft, Fortschritt und die Fähigkeit selbst etwas zu bewegen!

Ich danke auch meiner Frau, meinen Eltern und meinen Freunden, die mich immer unterstützt und gefördert haben.

INHALTSVERZEICHNIS

SYMBOL- UND ABKÜRZUNGSVERZEICHNIS

$\langle x, y \rangle_{\mathcal{R}}$	Das innere Produkt der Vektoren x und y bezüglich des Wörterbuchraums $\mathcal{R}$ (Definition 4.2)
bzip2	Ein populäres Kompressionsverfahren (Seite 30)
$C(x)$	Ein Kompressor (4.5)
$\mathcal{D}$	Ein Wörterbuch (Seite 68)
ETL	Extract-Transform-Load: Der Prozess des Einlesens von Daten aus unterschiedlichen Datenquellen in ein Datawarehouse (Seite 38)
$\mathcal{F}$	Ein beliebiger Vektorraum mit möglicherweise unterschiedlichen Vektorraumnormen (Kapitel 4.2). Wir werden mit diesem Vektorraum meist Funktionenräume assoziieren.
gzip	Eine populäre Implementierung des LZ77 Algorithmuses (Seite 48)
$K(x)$	Beschreibungskomplexität oder auch Kolmogorov Komplexität (Seite 20)
$K_A(y\|x)$	Bedingte Beschreibungskomplexität (Definition 2.3)
kontinuierlich	Eine Bezeichnung für eine Menge mit überabzählbarem Wertevorrat (Seite 8)
$d_{\mathrm{NCD}}(x, y)$	Die Normalisierte Kompressionsdistanz zwischen x und y (Definition 2.13)
NCD	Die Normalisierte Kompressionsdistanz (Definition 2.13)
$d_{\mathrm{NID}}(x, y)$	Die Normalisierte Informationsdistanz zwischen x und y (Definition 2.10)
NID	Die Normalisierte Informationsdistanz (Definition 2.10)
Ω_b	Die Menge der binären Zeichenketten (Seite 24)
RDS, $\mathcal{R}$	Redundante Wörterbuchräume (Kapitel 4.2).
$\mathcal{S}$	Ein beliebiger (Koordinaten-)Vektorraum mit möglicherweise unterschiedlichen Vektorraumnormen (Kapitel 4.2).

Abstract

Since the discovery of the Normalized Compression Distance (NCD) and its impressive abilities, Compression Based Pattern recognition has gained widespread recognition. This was sufficient for us to look under the hood of the NCD to gain deep and well founded insights into the area of what is up to now called Compression Based Pattern Recognition.

While looking at the NCD and its theoretical foundation, we found that there is a large gap between Kolmogorov complexity and the NCD. Cilibrasi und Vitànyi have successfully motivated the NCD by using the Kolmogorov complexity as a measure of similarity. Due to its incomputability, they failed to actually connect the NCD to its theoretical motivation and, besides that, the NCD largely lacks any support for continuous data. There is work on the NCD for image data but it takes no notice of the actual underlying continuous data.

In the course of this work we are researching the underlying principles of the NCD. Based on these findings we are developing algorithms which are focused solely on pattern recognition and represent the essence of Compression Based Pattern Recognition. We are extending these ideas to continuous pattern matching and thereby present an unifying theory which justifies the generallity of the denomination Compression Based Pattern Recognition. For all our findings we are also presenting detailed numerical experiments and formal mathematical proofs.

All in all we are giving the area of Compression Based Pattern Recognition a theoretical foundation as well as a new point of view.

Kurzfassung

Seit der Entwicklung der Normalisierten Kompressionsdistanz (in engl. Normalized Compression Distance – NCD) mitsamt ihren beeindruckenden Fähigkeiten, gewann kompressionsbasierte Mustererkennung immer mehr an Anerkennung. Die Arbeit beschäftigt sich daher mit der NCD und den damit verbundenen Theorien und Experimenten um genauer zu verstehen, was wirklich dieses Verfahren ausmacht und was für dessen Ergebnisse verantwortlich ist.

Augenblicklich gibt es noch eine große Lücke zwischen der praktischen Anwendung der kompressionsbasierten Mustererkennung und deren Fundament, der Kolmogorovkomplexität. Cilibrasi und Vitànyi haben zwar die NCD erfolgreich mit Hilfe der Kolmogorovkomplexität motiviert, jedoch aufgrund der Unberechenbarkeit dieses Komplexitätsmaßes noch keine echte Verbindung zwischen den beiden Methoden hergestellt. Des weiteren ist die NCD bisher auch weitgehend auf diskrete Daten beschränkt. Jede Anwendung auf kontinuierliche Daten behandelt diese, als wären sie diskret.

Im Rahmen dieser Arbeit werden die zugrunde liegenden Prinzipien der NCD untersucht. Dazu wird die Theorie und deren praktische Anwendung betrachtet, um aufbauend auf diesen Erkenntnissen, Mustererkennungsverfahren zu entwickeln, die die Essenz der kompressionsbasierten Mustererkennung darstellen. Dabei wird eine Theorie der kompressionsbasierten Mustererkennung, die die Ergebnisse der NCD erklären kann und auch inhärent kontinuierliche Methoden erlaubt, entwickelt. Zu dieser Theorie werden auch sowohl die notwendigen mathematischen Beweise als auch praktische Experimente präsentiert.

Alles in allem wird der kompressionsbasierten Mustererkennung ein theoretisches Fundament sowie ein neuer Blickwinkel gegeben.

KAPITEL

1 EINLEITUNG

Überblick

- Es gibt viele unterschiedliche Arten von Mustern. Jedes dieser Muster repräsentiert einen anderen Aspekt der Daten.
- Kompression sucht nach Mustern in Daten, um diese möglichst kompakt zu repräsentieren.
- Regularisierung sucht ebenfalls nach „kurzen“ Repräsentationen, um eindeutige Ergebnisse zu erhalten.
- Kompressionsbasierte Mustererkennung verbindet die Ideen von Kompression, Mustererkennung und Regularisierung.

Kompression und Mustererkennung haben viele Gemeinsamkeiten: Moderne Kompressionsverfahren versuchen, wiederkehrende Zeichenketten durch kürzere Referenzen zu ersetzen, und Mustererkennungsverfahren versuchen, in möglichst kompakten Beschreibungen eine ideale Repräsentation zu finden. In gewisser Weise handelt es sich um zwei Sichtweisen auf dieselbe Fragestellung:

> Welche Informationen sind wirklich relevant für die korrekte Wiedergabe eines Sachverhalts?

In der Mustererkennung sucht man, in Hinsicht auf eine möglichst gute Generalisierung, nach einer Beschreibung der Daten, die, trotz unvollständigen Wissens (man hat es ja zwangsläufig nur mit endlich vielen Repräsentanten einer potenziell unendlichen Menge zu tun), möglichst gut zu allen (auch den unbekannten) Daten passt. Hier kommt der Regularisierung eine zentrale Bedeutung zu. Wie wir später sehen werden,

wird bei dieser Methode auch nach einer möglichst kurzen Repräsentation gesucht – was uns wieder zur Kompression bringt.

In der Kompression sucht man nach einer möglichst kurzen Repräsentation von Daten, ohne dabei jedoch wesentliche Informationen zu vernachlässigen. Dabei wird versucht, wiederkehrende Muster zu erkennen und durch Referenzen zu ersetzen, also ganz klar ein Mustererkennungsschritt.

Diese Arbeit beschäftigt sich mit den Gemeinsamkeiten der Mustererkennung und der Kompression. Darauf aufbauend werden wir eine Theorie zur kompressionsbasierten Mustererkennung entwickeln und durch Experimente validieren. Ausgangspunkt hierfür wird die kritische Auseinandersetzung mit bestehenden Ansätzen – hauptsächlich den Arbeiten von Li et al. und Cilibrasi und Vitànyi zur Kolmogorovkomplexität und der normalisierten Kompressionsdistanz (NCD) – zur kompressionsbasierten Mustererkennung sein.

Bevor wir uns näher mit den einzelnen Themen der Arbeit beschäftigen, werden wir uns kurz einen Überblick über die Kernpunkte der Arbeit verschaffen.

Kolmogorovkomplexität Die Kolmogorovkomplexität hat sich etabliert als probates Mittel zur Behandlung komplexitätstheoretischer Fragestellungen. Gegen eine praktische Anwendung (wenn auch nur als Approximation) sprechen jedoch zahlreiche Aspekte, die im Rahmen dieser Arbeit erstmals in dieser Form aufgeführt und dokumentiert werden. Wir werden in Kapitel 2 Fakten darlegen und Beweise aufführen, die die Kluft zwischen den theoretischen Überlegungen und einer praktischen Anwendung veranschaulichen.

NCD Dass die NCD ein breites Anwendungsspektrum hat, wurde bereits in zahlreichen wissenschaftlichen Arbeiten dokumentiert. Wir werden hier eine weitere interessante Anwendung – die der Entitätenerkennung – darlegen. Neben der reinen Anwendung werden wir uns auch der Frage zuwenden, was genau die NCD so erfolgreich macht. In Rahmen dieser Arbeit werden wir eine Reihe von Experimenten dokumentieren, die Aufschluss geben über die Merkmale, die von der NCD betrachtet werden.

Räume über redundanten Wörterbüchern Die Menge aller minimaler Repräsentationen mit Hilfe von möglicherweise redundanten Wörterbücher wird in dieser Arbeit (Abschnitt 4.2) zu einem Vektorraum erweitert. Desweiteren werden wir ein Ähnlichkeitsmaß auf diesen Strukturen definieren, das uns den Vergleich von Daten bezüglich ihrer redundanten Repräsentation erlaubt.

Kompressionsbasierte Mustererkennung Kapitel 4 beschreibt eine generelle Theorie der kompressionsbasierten Mustererkennung, aufbauend auf den Ideen redundanter Wörterbücher und Räumen über diesen, mit deren Hilfe sich erstmals alle Ergebnisse aus Experimenten mit der NCD erklären lassen. Darüber hinaus werden wir in Kapitel 5 auch Experimente mit kontinuierlichen Daten durchführen.

1.1 Mustererkennung

Obwohl wir den Begriff Muster, sowohl im umgangssprachlichen Alltag als auch in unsere Tätigkeit als Wissenschaftler, sehr häufig verwenden, gibt es keine konkrete Definition für Muster. Was man genau darunter versteht, ergibt sich meist aus dem Zusammenhang. Wir werden Muster und die damit verbundene Mustererkennung als das Erkennen typischer Bereiche oder Aspekte in Daten verstehen. Dies können, je nach Datensatz, sich wiederholende Wörter in Texten oder aber auch häufig oder dominant auftretende Schwingungen in Bildern sein. Von sogenannten low-level Merkmalen, also jenen, die auf einer sehr niedrigen Stufe der Signalverarbeitung behandelt werden, bis hin zu abstrakten Strukturen, wie zum Beispiel syntaktisch oder gar semantisch zusammenhängenden Objekten, gibt es sehr viele verschiedene Arten von Mustern. Wir werden uns jetzt im Folgenden einige Beispiele ansehen.

Fangen wir an mit wiederkehrenden Symbolen oder Objekten in einer Datei. Eine solche besteht aus Zeichen. Diese könnten zum Beispiel Buchstaben in einer Textdatei, DNA-Tripel in einer DNA-Sequenz oder auch Bytes in einer Binärdatei sein. Da es jeweils davon nur eine endliche Anzahl gibt, werden diese auch als diskret bezeichnet. Bei gängigen Computern gibt es üblicherweise 256 unterschiedliche Zeichen. Betrachtet man diese separat, also unabhängig von ihrem Kontext, so kann man häufig wiederkehrende Zeichen erkennen. Hier nun ein sehr praxisnahes Beispiel, wie solche wiederkehrenden Zeichen oder Objekte Muster darstellen.

Beispiel 1.1 (Wiederkehrende Zeichen)**.** *Betrachten wir eine Datei, die den Besuch von Kunden in einem Restaurant dokumentiert. Immer wenn ein Gast das Restaurant betritt, wird die ihm zugewiesene Nummer in einer Datei gespeichert. Wiederkehrende Gäste erhalten immer dieselbe Nummer. Ein Muster wäre nun, dass ein Gast häufiger als andere Gäste in dem Restaurant isst. Dieses Muster bezeichnet man üblicherweise als "Stammgast".*

Für diese Art von Muster gibt es viele weitere Anwendungsfälle, so zum Beispiel bei der Bestimmung relevanter Zeichen oder der Bestimmung der Herkunft von Daten. Wesentlich mehr Information erhält man jedoch, wenn man nach größern, komplexeren Strukturen sucht. Hier bietet es sich an, nach einer Abfolge von Zeichen zu suchen.

Es ist eher selten, dass das Auftreten von Zeichen unkorreliert ist. Auf ein bestimmtes Zeichen folgt mit hoher Wahrscheinlichkeit ein anderes bestimmtes Zeichen. In Texten zum Beispiel folgt auf ein 'u' mit hoher Wahrscheinlicheit ein 'n' wie in 'und' oder 'Hoffnung'. Solche und noch komplexere Zusammenhänge, wie zum Beispiel das Erkennen von Wörtern als Zeichenfolgen von Buchstaben zwischen Leerzeichen, ist auch eine Art von Mustererkennung. Sie erfordert jedoch deutlich mehr Daten als die Erkennung von wiederkehrenden Zeichen.

Beispiel 1.2 (Wiederkehrende Zeichenketten)**.** *Wenn wir bei unserem Stammgastbeispiel bleiben, so könnten wir vielleicht feststellen, dass immer, wenn ein bestimmter Gast das Restaurant betritt, auch ganz bestimmte weitere Gäste dort essen. Man erkennt also Gruppen. Diese würde man nicht erkennen, wenn man nur nach der Anzahl*

der Besuche der Gäste sucht, man muss also auch die sequenziellen Zusammenhänge betrachten.

Zeichen sind ein Aspekt *diskreter* Daten. Als solche werden wir Daten bezeichnen, die eine abzählbare Menge an möglichen Werten haben. Diese sind per Definition klar und ohne Zwischenschritte voneinander zu unterscheiden. *Kontinuierliche* Daten haben im Gegensatz dazu einen überabzählbaren Wertevorrat, zwischen jedem Wert gibt es eine unendliche Anzahl von Zwischenschritten. In der Welt der Computer lässt sich dies jedoch nicht realisieren, so dass wir auch solche Daten als kontinuierlich bezeichnen, die nur abgetastete Repräsentation kontinuierlicher Funktionen sind. Beispiele hierfür sind Bild- oder Geräuschdaten, die auf einem Computer als abgetastete Signale (Funktionen) gespeichert werden. Diskrete Daten sind inhärent diskret, im Gegensatz zu kontinuierlichen Daten – die im Computer ja auch diskret abgespeichert werden – also Daten, deren exakte Repräsentation ebenfalls diskret ist. Beispiele für inhärent diskrete Daten sind Text oder DNA, aber auch Binärdateien wie Maschinencode oder Netzwerkpakete. Gerade bei der Handhabung kontinuierlicher Daten ist jedoch eine Betrachtung der einzelnen Symbole oder deren Abfolgen nicht hilfreich. Hier sucht man eher nach Ähnlichkeiten zu existierenden Signalen. Dieser Blickwinkel und die Sichtweise von Funktionen als Elemente eines Vektorraums helfen, das nachfolgende Beispiel für eine Art Muster zu verstehen.

Vektoren in einem Vektorraum werden üblicherweise als gewichtete Linearkombination von Basiselementen gesehen. Der Vektor selbst wird dann mit der Gewichtung gleichgesetzt, was im Fall der kanonischen Basis auch durchaus seine Berechtigung hat[1]. Betrachtet man nun zwei Vektoren, reicht es aus, die Gewichtungen zu vergleichen. Eine Gewichtung, die in allen Vektoren enthalten ist, nennen wir auch in diesem Fall Muster.

Beispiel 1.3 (Ähnlichkeit zu markanten Funktionen)**.** *Betrachten wir in diesem Beispiel nicht die Gäste des Restaurants, sondern die Anzahl der Gäste, die sich über den Tag verteilt in dem Restaurant befinden. Wenn wir nun die Abstände zwischen den Messungen gegen null gehen lassen so erhalten wir auch reelle Anzahlen (also zum Beispiel $\sqrt{2}$ Gäste, wenn von zwei Gästen die das Restaurant verlassen der Zweite noch nicht ganz zur Tür hinaus ist). Um unsere Daten dann darzustellen, benötigen wir eine Funktion mit kontinuierlichem Zeitindex und kontinuierlichem Wertebereich. Für diese Funktion einen geeigneten Repräsentanten zu finden, ist gar nicht so leicht, da es unendlich viele Möglichkeiten gibt. Abbildung 1.1 zeigt eine Variante mit gaußschen Glockenkurven unter Verzicht auf unwichtige*[2] *Informationen. Auch diese Funktion kann markante Aspekte, haben die wir als Muster sehen können. Diesmal jedoch sind sie Teil der Repräsentation und nicht mehr der Daten.*

Wichtig bei der Unterscheidung des Letzten von den vorangegangenen Mustern ist der Aspekt, dass dieses Muster weder ein klar definierten Anfang noch ein eben

[1] Dass es hier auch andere Fälle geben kann, werden wir bei der Behandlung redundanter Wörterbücher in Kapitel 4.2 sehen.

[2] Das Unwichtige wird in diesem Fall durch die Repräsentation definiert: Aspekte einer Funktion, die sich nur schlecht darstellen lassen, gelten als unwichtig (vgl. Mallat 2009, Kapitel 10.4).

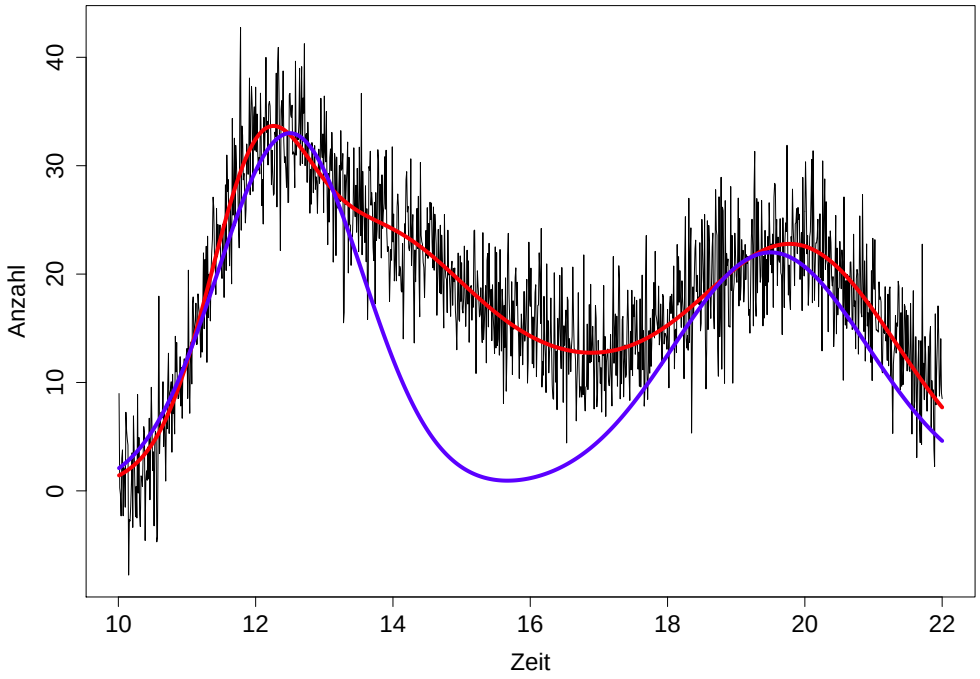


Abbildung 1.1: Anzahl der Gäste pro Zeit in einer Gaststätte. Die schwarze Kurve entspricht der gemessenen Anzahl, die rote Kurve einer Repräsentation – ein anderer möglicher Begriff für Repräsentation ist Approximation – mit Hilfe gaußscher Glockenkurven (wobei weniger wichtige Informationen entfernt wurden), und die blaue Kurve steht für eine weitere mögliche Repräsentation mit Gaußglocken als Basis. Der wesentliche Unterschied zwischen den beiden Repräsentationen liegt in dem Verhältnis, in dem die Anzahl der Basiselemente zu dem Approximationsfehler steht. In Kapitel 4 wird dieses Verhältnis ausführlich behandelt.

solches Ende hat. Es kann zu jedem Teil des Signals beitragen. Ein Beispiel hierfür wäre eine markante niederfrequente Schwingung, die ein Signal überlagert, wie bei Herzerkrankungen als tiefes Rauschen während der Herzschläge hörbar.

Ein Ziel dieser Arbeit ist es, Methoden zu entwickeln, die es erlauben, solche Muster zu erkennen, als relevant und wichtig zu identifizieren und wiederzuerkennen, also auch in neuen Daten wahrzunehmen. Für das erstmalige Erkennen bedienen wir uns den Methoden der Kompression.

1.2 Kompression

Kompression hat die Aufgabe, Daten aus einer – möglicherweise ungünstigen – Repräsentation in eine kompaktere zu überführen. Dabei geht es darum, Muster zu erkennen und durch platzsparende Alternativen zu ersetzen. Diese Muster können, wie im Fall von LZ77, sich wiederholende Sequenzen oder, wie im Fall von JPEG, eine domi-

nante Frequenz sein. Man unterscheidet zwischen verlustfreien und verlustbehafteten Verfahren. Verlustfreie Verfahren transformieren die Eingabe so, dass eine exakte Rekonstruktion wieder möglich ist. Die verlustbehafteten Verfahren verändern die Daten dagegen so, dass möglichst nur die relevanten Informationen erhalten bleiben.

Wie wir in Kapitel 3.1 sehen werden, eignen sich verlustfreie Verfahren besonders gut für diskrete Daten. Dabei kommt gerade den Methoden von Ziv und Lempel eine besondere Bedeutung zu. Diese suchen nach wiederkehrenden Zeichenketten, die zur Kompression herangezogen werden – was in der resultierenden Datei zu einer Abfolge unkorrelierter Daten führt. Die gefundenen Zeichenketten helfen jedoch nicht nur bei der Kompression, sie sind auch repräsentativ für die Daten. Wenn man die gefundenen Zeichenketten von zwei komprimierten Dateien vergleicht, erhält man ein Ähnlichkeitsmaß für die Dateien.

Betrachtet man kontinuierliche Daten, so ist es nicht mehr so einfach, wiederkehrende Zeichenfolgen oder deren kontinuierliches Analogon, repräsentative Funktionen, zu finden. Hier beschränkt man sich darauf, Transformationen durchzuführen, die wesentliche Aspekte der Daten hervorheben und alle unwesentlichen Aspekte einfach zu ignorieren. Die wohl populärsten Beispiele hierfür sind der MP3-Kompressor und das JPEG-Verfahren (beide zu finden in Chakrabarti 2009). JPEG konzentriert sich darauf, niederfrequente Daten möglichst gut wiederzugeben, was durch die Frequenzverteilung in natürlichen Bildern motiviert ist. Ein neuer Ansatz zur kompakten Repräsentation von kontinuierlichen Signalen ist die Darstellung als dünn besetzte Linearkombination redundanter Funktionen. Hierbei werden nicht mehr nur Basiselemente herangezogen, sondern auch Vektoren, die linear abhängig sind und somit nicht mehr so einfach zu handhaben sind. Auch das Ergebnis, zwar ein Vektor jedoch nicht mehr zwangsläufig eindeutig, lässt sich nicht mehr so einfach mit anderen Vektoren vergleichen. Wir werden jedoch auch hierfür ein kompressionsbasiertes Ähnlichkeitsmaß entwickeln.

1.3 Kompressionsbasierte Mustererkennung

Das Interessante an einem Ähnlichkeitsmaß, das man durch Kompressionsverfahren erhält, ist jedoch nicht die Tatsache, dass man damit Dateien vergleichen kann. Es ist der Fakt, dass die Ähnlichkeit erlernt ist. Sie wird nicht im Vorhinein durch einen Trainer vorgegeben, sondern im Rahmen des Mustererkennungsprozesses selbst erlernt.

Cilibrasi und Vitànyi und zahlreiche weitere Wissenschaftler führen diese Fähigkeit auf die Urform aller Kompressionsverfahren, die Kolmogorovkomplexität, zurück. Diese nichtberechenbare Funktion hat, unter anderem aufgrund ihrer Nichtberechenbarkeit, einige sehr interessante Eigenschaften, mit deren Hilfe man jedes Mustererkennungsproblem lösen könnte. Wie wir jedoch sehen werden, sind diese Eigenschaften eher theoretischer Natur, in der Praxis lassen sie sich nicht umsetzen. Aussagen, wie die von Keogh et al. (2004),

> However, as a practical matter, they [methods based on the Kolmogorov Complexity] can be implemented using any off-the-shelf compression algorithm with the addition of just a dozen or so lines of code.

sind daher mit Vorsicht zu genießen, da sie suggerieren, dass sich die herausragenden Eigenschaften der Kolmogorovkomplexität in jedem beliebigen Kompressor wiederfinden.

Wir werden uns mehr an die Ideen von Sculley und Brodley (2006) halten. Das bedeutet, dass wir die Arbeit mit Kompressionsverfahren, die viele interessante Ergebnisse generiert, näher unter die Lupe nehmen werden, ohne uns dabei allzu sehr auf die Kolmogorovkomplexität als mögliche Erklärung zu versteifen. Es lassen sich mit Hilfe von Kompressionsverfahren zum Beispiel Texte ihren Autoren zuordnen oder Säugetiere anhand ihrer Verwandtschaftsbeziehung gruppieren. All das gibt Grund zur Annahme, dass hinter der kompressionsbasierten Mustererkennung doch mehr steckt als die Kritik an der Nichtberechenbarkeit der Kolmogorovkomplexität vermuten lässt.

Ausgehend von den kontinuierlichen Kompressionsverfahren ist zu prüfen, ob die Überlegungen zu kompressionsbasierten Verfahren auch auf kontinuierliche Daten übertragbar sind. Betrachtet man die Kolmogorovkomplexität, die ja nach kürzesten Turingmaschinen sucht, wird man schnell enttäuscht werden. Löst man sich jedoch von dieser Motivation und betrachtet die diskreten Kompressionsverfahren als Methoden zur Wörterbuchgenerierung (was ja auch sehr nahe am eigentlichen Anliegen von Ziv und Lempel liegt), so kommt man schell zu der Sichtweise, dass diese Verfahren zur Mustererkennung Wörterbücher vergleichen. Dann ist auch der Sprung zu kontinuierlichen Verfahren nicht mehr weit. Auch hier lassen sich Wörterbücher vergleichen. Gerade die Methoden, die um die Arbeit mit redundanten Wörterbüchern (Redundant Dictionaries und Sparse Coding) kreisen, liefern hier eine wahrhafte Fülle an Analogien, die sich umsetzen lassen.

1.4 Regularisierung

Ein letzter Aspekt der kompressionsbasierten Mustererkennung ist die Regularisierung. In vielen Anwendungsfällen und in zahlreichen theoretischen Arbeiten hat sich die Überzeugung durchgesetzt, dass man versuchen sollte, nicht nur den Fehler (den man bei der Mustererkennung zwangsläufig macht) klein zu halten, sondern auch die Komplexität des zugrunde liegenden Modells sollte gering sein. Dies geht auf Überlegungen von Tichonov und Arsenin (1977) zurück. Dabei ging es um die Lösung von sogenannten ill-posed Problems. Diese wird mit Hilfe einer Suche nach möglichst „kurzen“ Lösungsvektoren eindeutig bestimmt. Tichonov und Arsenin ermöglichten auf diese Weise, Fragestellungen eindeutig zu beantworten, die vorher nicht beantwortbar waren, eine Problemstellung, die auch in der Mustererkennung immer wieder auftritt.

Dieser Ansatz wurde von Vapnik (1998) so interpretiert, dass man durch die Suche nach einer möglichst kurzen Lösung den strukturellen Fehler, also den Fehler, den man

macht, wenn man eine größere Komplexität der Mustererkennungsaufgabe annehmen würde, minimiert. Unter Komplexität wird hierbei oft die Anzahl der beteiligten Variablen verstanden. Je mehr Variablen beteiligt sind, umso komplexer ist das Problem. Unter dem strukturellen Fehler versteht man nun den Fehler, der Variablen zur Lösung eines Problems heranzieht, die dafür gar nicht benötigt werden. Formal wird dieser Fehler mit Hilfe eines konvexen Maßes bestimmt. Meistens wird hierfür die ℓ^2 oder ℓ^1 Norm des Koeffizientenvektors herangezogen (vgl. Kapitel 4).

Kompressionsbasierte Mustererkennung versucht ebenfalls eine Repräsentation für Daten zu finden, die möglichst kurz ist. Dabei wird, wie bei der Regularisierung auch, ein Kompromiss zwischen exakter Rekonstruktion und länge der Repräsentation gesucht. Der Begriff *kurz* bezieht sich in diesem Kontext auf eine Zeichenkette mit möglichst wenig Zeichen, kann aber auch als ein Vektor mit möglichst wenig Einträgen interpretiert werden (wie wir ebenfalls in Kapitel 4 sehen werden). Der Vergleich zweier komprimierter Zeichenketten findet dann anhand dieser gefundenen Repräsentation statt.

KAPITEL

2 THEORIE DER NCD

Überblick

- Das Konzept der Universalität der Kolmogorovkomplexität ist, von einem praktischen Standpunkt aus gesehen, nur für beschränkte Mengen und für sehr lange Zeichenketten von Bedeutung (Korollar 2.1).
- Die Kolmogorovkomplexität nutzt nur bei sehr wenigen Zeichenketten Informationen aus der zur Verfügung stehenden Zeichenkette (Satz 2.3).
- Wir definieren die Normalisierte Informationsdistanz (Def. 2.10) und die Normalisierte Kompressionsdistanz (Def. 2.13).
- Die NCD ist allgemein, universell und robust gegen Rauschen (Absch. 2.3.1).

Die Normalized Compression Distance (NCD) wird in den Arbeiten von Li et al. und Cilibrasi und Vitànyi direkt durch die Kolmogorovkomplexität motiviert. Durch die Analogie der Kompression ist diese Vermutung auch naheliegend, ob sie jedoch auch in der Realität vertretbar ist, werden wir in diesem Kapitel untersuchen.

2.1 Kolmogorovkomplexität

In diesem Abschnitt beschäftigen wir uns mit der Kolmogorovkomplexität. Dabei handelt es sich um ein absolutes Informationsmaß. Shannon betrachtet den Informationsgehalt einer Nachricht abhängig von dem Kontext, in dem sie sich befindet (Shannon 1948). Etwas konkreter ausgedrückt bedeutet dies, dass wir den Zeichensatz kennen,

der für die Nachricht verwendet wird, ebenso wie die Auftrittswahrscheinlichkeit von jedem Zeichen. Der Informationsgehalt einer Nachricht ergibt sich dann aus den verwendeten Zeichen und deren Wahrscheinlichkeiten. Nachrichten mit sehr wahrscheinlichen Zeichen haben einen geringeren Informationsgehalt als solche mit hauptsächlich unwahrscheinlichen Zeichen. Dies kann zu vordergründig merkwürdigen Beispielen führen wie z.B. dem von Sayood (2000):

> The barking of a dog during a burglary is a high-probability event and, therefore, does not contain too much information. However, if the dog did not bark during a burglary, this is a low probability event and contains a lot of information.

Da wir ja bereits wissen, dass ein Einbruch stattfindet (das ist der Kontext und definiert den Zeichenvorrat samt Wahrscheinlichkeiten), ist das Ausbleiben des Hundebellens (ein sehr wahrscheinliches Zeichen) tatsächlich von deutlich größerer Information, als dessen Ertönen (was wohl auch Sayood und Sherlock Holmes bewußt war – Silberstern von Arthur Conan Doyle). Auch das üblicherweise vorgebrachte Beispiel einer vollkommen willkürlichen Zeichenkette (die ja intuitiv keine Information enthält) muss im Kontext der möglichen Nachrichten betrachtet werden und dort kommt exakt dieser Nachricht (von der man ja nicht weiß, dass sie rein zufällig ist) eine sehr hohe Bedeutung zu.

Kolmogorovs Absicht war es jedoch nicht, den Informationsgehalt von Nachrichten im Rahmen ihres Kontextes zu betrachten, sondern diesen absolut – also unabhängig von der zugrundeliegenden Menge aller möglicher Nachrichten – zu bestimmen. Dies würde bedeuten, dass Goethes Faust mehr Information trägt als eine gleichlange Folge eines einzigen Zeichens – was auch klar zu sein scheint. Goethes Faust würde wahrscheinlich auch mehr Information beinhalten als die entsprechende Abfolge von Zeichen in der Zahl π (vgl. „The Infinite Monkey Theorem“). Nicht nur deshalb gibt es auch erhebliche Zweifel an der Existenz eines absoluten Informationsmaßes. Lempel und Ziv äußerten sich zum Beispiel sehr deutlich gegen ein solches Maß (Lempel und Ziv 1976) und auch die Nichtberechenbarkeit (als Beweis dafür steht Satz 2.5) lässt erhebliche Zweifel an dessen Aussagekraft. Beispiele gegen die Existenz, die intuitiv einleuchten, wären Sätze wie John F. Kennedys *Ich bin ein Berliner*, oder die Zeichen π und e. Diesen einen Informationsgehalt zuzuweisen, dürfte sehr schwierig sein, da es neben der Zeichenkette selbst noch eine tiefere Bedeutung gibt. Noch deutlicher wird es bei dem Symbol für den Imaginärteil komplexer Zahlen i. Wie verhält sich dessen Informationsgehalt zu dem neunten Zeichen im Alphabet i? Nach Kolmogorov wäre auch der Informationsgehalt der Zahl 10^{21} ähnlich groß wie der Informationsgehalt 10^{22}, von beiden jedoch deutlich geringer als der von $501,436,928$, was nun wiederum nicht intuitiv klar ist. Ich teile die hier angedeutete Skepsis gegenüber der Existenz einen solchen Maßes. Sie war auch die Motivation zu dieser Arbeit. Um jedoch die Diskussion auf eine solide Grundlage zu stellen, möchte ich hier einen kurzen Exkurs in den Bereich der Algorithmischen Informationstheorie starten. Wir werden uns dabei sehr eng an das Buch von Li und Vitànyi halten.

Bevor wir jedoch mit der Definition der Kolmogorovkomplexität starten, ist es sinnvoll, ein paar grundlegende Aspekte zu klären. Unsere folgenden Betrachtungen stützen sich hauptsächlich auf die Monographien von Lüneburg (2002), Kleene (1971) und Rogers (1967). Einige der folgenden Sätze werden ohne Beweis vorgestellt, dieser findet sich dann in der zitierten Literatur.

Im Folgenden werden wir davon ausgehen, dass wir den Elementen einer Menge $X = \{x\}$ eine natürliche Zahl $n(x) \in \mathbf{N}$ zuweisen können. Dies setzt voraus, dass X abzählbar ist. Die zugewiesene Zahl $n(x)$, manchmal auch mit n abgekürzt, lässt sich in Form einer endlichen Sequenz von Nullen und Einsen repräsentieren. Die Länge dieser Sequenz ist gegeben durch die Längenfunktion $l(x)$.

Definition 2.1 (Längenfunktion – Li et al. (2004))**.** *Eine Längenfunktion $l : \mathbf{N} \mapsto \mathbf{N}$ ordnet jeder natürlichen Zahl ihre lexikografische Position (innerhalb der sortierten Liste aller natürlicher Zahlen) zu.*

Für die Konkatenation x, y zweier natürlicher Zahlen werden wir annehmen, dass

$$l(x, y) \leq C_x + l(y) \tag{2.1}$$

wobei die Konstante C_x nur von x abhängig ist. Zur besseren Veranschaulichung hier ein kurzes Beispiel:

Beispiel 2.1. *Die lexikogafische Ordnung der natürlichen Zahlen als Binärzeichenketten:*

$$(\epsilon, 0), (0, 1), (1, 2), (10, 3), (11, 4), (100, 5), \ldots.$$

Hier wäre dann zum Beispiel $l(11) = 4$.

Des Weiteren werden wir uns mit partiell rekursiven Funktionen beschäftigen. Diese und ganz besonders deren Nummerierung stellen einen zentralen Aspekt der Kolmogorovkomplexitätstheorie dar.

Definition 2.2. *Eine partiell rekursive Funktion φ mit k Variablen ist eine Abbildung, die auf einer Teilmenge von $\mathbf{N}^k$ definiert ist, Werte in $\mathbf{N}$ hat und deren Graph rekursiv abzählbar ist.*

Um in geeigneten Fällen die Nähe zu Maschinen und Programmen zu verdeutlichen, werden wir manchmal die Funktion φ als Computer, die erste Variable als Programm mit p und die zweite Variable als Daten mit x bezeichnen. Damit erhalten wir auch die Schreibweise $p \in P_\varphi$, $x \in X_\varphi$, $y \in Y_\varphi$ und $(x, y) \in D_\varphi$ falls $\varphi(p, x) = y$. Dabei repräsentieren P_φ und X_φ die Domänen von φ und Y_φ deren Wertemenge.

Jetzt, da wir alle relevanten Voraussetzungen dargestellt haben, können wir uns der bedingten Beschreibungskomplexität zuwenden. Sie stellt die Basis für alle weiteren Komplexitätsbetrachtungen dar und führt auch direkt zu dem, was üblicherweise unter Kolmogorovkomplexität oder algorithmischer Komplexität verstanden wird. Wir halten uns hierbei an die Definition von Kolmogorov (1993, chapter 10).

Definition 2.3 (Bedingte Beschreibungskomplexität – Kolmogorov (1993))**.** *Die bedingte Beschreibungskomplexität $K(y|x)$ von y gegeben x und die partiell rekursive Funktion φ, ist definiert durch*

$$K_\varphi(y|x) = \min\{l(p)|\varphi(p, x) = y, \}.$$

Falls φ nicht für p, x und y definiert ist, es also kein p gibt sodass $\varphi(p, x) = y$ ist, so gilt $K_\varphi(y|x) = \infty$.

Hier fällt, wie schon eingangs angedeutet, die Interpretation der ersten Variable p als „Programm“ und φ als „Computer“ auf. Des weiteren ist die Länge von p natürlich maßgeblich von φ abhängig. Jeder Programmierer weiß, wie sich mit Hilfe domänenspezifischer Programmiersprachen manche Probleme leicht lösen lassen, die mit anderen Sprachen sehr mühsam zu lösen sind. Für φ und p bedeutet dies, dass – abhängig von der Wahl der Funktion – die Eingabe zur Generierung einer speziellen Zeichenkette entweder sehr lang oder auch sehr kurz sein kann. Um hier eine einheitliche Herangehensweise zu schaffen, also die Länge von p unabhängig von der Wahl der Maschine φ zu halten, betrachten wir die Menge aller möglichen Funktionen und, in einem zweiten Schritt, sogenannte universelle Funktionen.

Definition 2.4 (Gödelnummer – Rogers (1967))**.** *Sei φ_n ein Element der Menge aller partiell rekursiven Funktionen, wobei n Index oder Gödelnummer von φ_n genannt wird.*

Hierbei sind wir stillschweigend davon ausgegangen, dass die Menge aller partiell rekursiven Funktionen abzählbar ist. Ein Beweis dafür findet sich bei Rogers (1967, §1.8). In diesem Zusammenhang sollte auch erwähnt werden, dass die Nummerierung keinesfalls eindeutig ist. Es gibt sogar unendlich viele mögliche Nummerierungen (Rogers 1967, chapter 4) worauf wir im weiteren Verlauf aufbauen werden. Falls wir jedoch eine dieser Nummerierungen festhalten, so stellt sich die Frage, ob es nicht eine Funktion gibt, die abhängig von einer weiteren Variable, andere Funktionen simuliert.

Satz 2.1 (Universelle Funktionen – Rogers (1967))**.** *Es existiert ein Index n, sodass für alle m und alle x*

$$\varphi_n(m, x) = \varphi_m(x)$$

gilt, falls $\varphi_m(x)$ definiert ist. Falls $\varphi_m(x)$ nicht definiert ist, so ist auch $\varphi_n(m, x)$ nicht definiert.

Um konsistent mit den Arbeiten von Kolmogorov und der üblichen Notation zu sein, werden wir anstelle von φ_n die verwendete universelle partiell rekursive Funktion mit $A(m, x)$ bezeichnen. In manchen Fällen wird auch $A(m, p, x) = \varphi_m(p, x)$ geschrieben, um die Existenz einer weiteren Variable p (in diesem Kontext auch oft Programm genannt) herauszustellen.

Wir wissen also nun, dass es eine universelle Funktion gibt, mit der sich andere Funktionen simulieren lassen. Für die Kolmogorovkomplexität bedeutet dies, dass wir in der Lage sind, ein Komplexitätsmaß anzugeben, welches sich so verhalten kann, als wäre es nicht an eine feste Funktion gebunden. Diese Freiheit hat jedoch auch einen Nachteil. Sie verlängert, bezüglich einer Funktion φ, die benötigte Eingabe um die Größe C_φ. Dies bedeutet, dass die Länge der Eingabe nicht nur von der Eingabe selbst abhängt, sondern auch von der gewählten universellen Funktion.

Satz 2.2 (Kolmogorov (1993))**.** *Es existiert eine partiell rekursive Funktion $A(p, x)$, sodass für jede andere partiell rekursive Funktion $\varphi(p, x)$ die Ungleichung*

$$K_A(y|x) \leq K_\varphi(y|x) + C_\varphi$$

gilt. Dabei ist C_φ weder von x noch von y abhängig.

Kolmogorov konnte zeigen, dass C_φ nur von der Funktion φ abhängt. Einzig die Nummerierung der partiell rekursiven Funktionen bestimmt die Größe dieser Konstante. Hier nun der Beweis, wobei wir uns an dem Weg von Kolmogorov orientieren werden.

Beweis. Der Beweis basiert auf der Existenz einer universellen partiell rekursiven Funktion $\varphi_n(m, x)$. Die von uns geforderte Funktion können wir durch

$$A((m, q), x) = \varphi_n(m, (q, x))$$

definieren. Aufgrund der Universalität erhalten wir

$$y = \varphi_m(q, x) = \varphi_n(m, (q, x))$$

und damit

$$A((m, p), x) = y, l(m, p) \leq l(p) + C_{\varphi_m}$$

□

An diesem Punkt erwähnt Kolmogorov, dass für eine solche Funktion A die algorithmische Komplexität $K_A(y|x)$ für alle Paare (x, y) endlich ist. Was uns jedoch viel mehr interessiert, ist, für welche Funktionen φ die Konstante C_φ klein ist. Diese Konstante repräsentiert eine Art „Größenabschätzung“ und daher ist es nur naheliegend, zu hoffen, dass C_φ für möglichst viele Funktionen sehr klein ist. Wesentlich für unsere weitere Arbeit ist auch die Tatsache, dass C_φ unabhängig von der Ein- und Ausgabe nur an die Funktion φ gebunden ist. Das folgende Korollar folgt mehr oder weniger direkt aus der Definition der universellen Kolmogorovkomplexität.

Korolar 2.1. *Die Konstante C_φ aus Satz 2.2 wird beliebig groß für mehr als endlich viele Funktionen φ.*

Beweis. Nach der Definition einer universellen partiell rekursiven Funktion (Rogers 1967, 1.8 Theorem IV) ist $A((n, p), x)$ für alle Werte definiert für die auch φ definiert ist. Wir betrachten nun den Fall, dass die Konstante C_φ durch eine obere Schranke C begrenzt wird ($C_\varphi < C < \infty$). Aufgrund der Definition von A ist

$$l(n, p) - l(p) \leq C_\varphi < C \qquad \text{(wobei } l(p) < l(n, p)\text{)}. \tag{2.2}$$

Hierbei hängt C_φ nur von φ ab. Wir können also für p jeden beliebigen Wert annehmen, ohne dass dies Einfluss auf C_φ hat. Zur einfacheren Anschauung wählen wir $p = \epsilon$ (wie in Beispiel 2.1) wodurch wir

$$l(n, \epsilon) - l(\epsilon) = l(n) \leq C_\varphi < C. \tag{2.3}$$

erhalten. □

Eine weitere interessante Aussage liefert in diesem Zusammenhang auch das Padding-Lemma (Schnorr 1974). Rogers (1967, page 23) interpretiert die gerade gewonnenen Erkenntnisse in folgender Weise:

> ... there is a critical degree of "mechanical complexity„ beyond which all further complexity can be absorbed into increased size of program and increased use of memory storage.

Was Rogers damit aussagt, ist, dass man fast jedes Problem das man mit einem Computer lösen kann, auch mit relativ simplen Computern lösen kann, solange man nur genügend Rechenleistung (physikalischen Speicher, CPU-Leistung, etc.) zur Verfügung hat. Falls es eine Funktion – in einem anderen Kontext einen Computer – gibt, die das Problem lösen kann, so können wir diesen einfach simulieren und erhalten die gewünschte Lösung. Korollar 2.1 zeigt, dass, gegeben eine partiell rekursive Funktion, der Speicherbedarf zur Simulation dieser Funktion mit Hilfe einer universellen Funktion nur für sehr wenige Funktionen klein ist. Für Komplexitätsbetrachtungen ist die Konstante C_φ als Approximationsfehler zu interpretieren. Falls wir also eine konkrete Funktion φ durch eine universelle ersetzen, laufen wir Gefahr, einen Fehler der Größe von C_φ zu machen. Dieser ist leider nur für sehr wenige Funktionen klein. Wichtig für unsere weiteren Betrachtungen ist die Tatsache, dass dieser Approximationsfehler nur von der Nummerierung der partiell rekursiven Funktionen abhängt. Welchen Einfluss diese Nummerierung hat, werden wir mit den nächsten beiden Beispielen verdeutlichen. Wir betrachten dabei endliche Mengen, was jedoch für den praktischen Einsatz keine Einschränkung darstellt.

Beispiel 2.2. *Gegeben sei eine Nummerierung der Menge der partiell rekursiven Funktionen, die jeder auf einer endlichen Menge X definierten, konstanten Funktion $\phi_y = y$, wobei y aus einer endlichen Teilmenge $Y = \{1...n\}$ der natürlichen Zahlen kommt, den Wert ihrer Rückgabe als Nummer zuweist. Alle weiteren, also nicht konstanten, Funktionen werden anschließend angefügt. Für diese Nummerierung ergibt die Bedingte Beschreibungskomplexität $K_A(y|x)$ zweier Objekte $x \in X$ und $y \in Y$ immer den Wert y.*

Beispiel 2.3. *Betrachten wir nun eine ähnliche Menge nur mit dem Unterschied, dass die konstanten Funktionen mit geradem Funktionswert $y \in Y$ (wieder die selbe endlichen Menge $Y = \{1...n\}$) der Reihe nach vom ersten Element an durchnumeriert sind und die Ungeraden hinten angehängt werden. Nun erhalten wir für alle geraden $y \in Y$ vernünftige Werte, für alle ungeraden $y \in Y$ jedoch unangemessen hohe Komplexitätswerte.*

Einen anderen Blickwinkel auf dieses Problem liefert (Cover und Thomas 2006, Kapitel 12, Seite 428) im Kontext der Kompression.

> We can then ask the question: How well can we compress a sequence? If we do not put any restrictions on the class of algorithms, we get a meaningless answer – there always exists a function that compresses a particular sequence to one bit while leaving every other sequence uncompressed.

In beiden Fällen könnte man nun argumentieren, dass dies Extrembeispiele sind und im Großen und Ganzen die Kolmogorovkomplexität nur die Funktionen auswählt, die einen vernünftigen Kompromiss zwischen Gödelnummer und Programmlänge darstellen. Solch ein Kompromiss dürfte jedoch nur in Ausnahmefällen auf konstante Funktionen zurückgreifen (diese stellen immerhin ein Extremum zwischen Gödelnummer und Programmlänge dar). Wie wir nun zeigen werden, stellen konstante (und quasiskonstante Funktionen) eher die Regel als eine Ausnahme dar.

Definition 2.5 (Quasikonstante Funktionen)**.** *Eine Funktion wird quasikonstant genannt, falls sie für mehr als endlich viele Argumentwerte denselben Funktionswert hat.*

Ausgehend von dieser Definition ist jede konstante Funktion auch quasikonstant. Des Weiteren beinhalten sowohl konstante als auch quasikonstante Funktionen nur sehr wenig komplexitätsbezogene Information. Die Definition der Kolmogorovkomplexität basiert auf der Annahme, dass die Komplexität eines Objekts gegeben ist durch das Programm, das es erzeugt. Für quasikonstante Funktionen ist dieses Programm jedoch sehr kurz, egal wie das Objekt selbst aussieht. Diese Überlegung führt uns zu folgender Definition und den darauf folgenden Satz.

Definition 2.6 (Endliche Flexibilität)**.** *Eine partiell rekursive Funktion A besitzt eine endliche Flexibilität wenn die Mengen*

$$F_A^x = \{(p, y) \textit{ mit } p \in P_A, y \in Y_A \, A(p, x) = y\}$$

und

$$F_A^y = \{(p, x) \textit{ mit } p \in P_A, x \in X_A \, A(p, x) = y\}$$

abzählbar unendlich für alle x und y sind.

Satz 2.3. *Sei die Funktion A partiell rekursiv und mit endlicher Flexibilität, dann existiert für jedes y eine quasikonstante Funktion, die von der Kolmogorovkomplexität K_A für mehr als endlich viele x Werte ausgewählt wird.*

Beweis. Der Beweis leitet sich direkt aus der Definition der Flexibilität ab. Zu jedem y gibt es eine partiell rekursive konstante Funktion mit Index n' (wir werden im Folgenden Funktionen φ_n und deren Indizes n synonym verwenden), sodass $A(n', \epsilon, x) = y$ für alle x ist. Da K_A immer die Funktionen auswählt für die $l(n, p)$ minimal ist gilt $l(n, p) \leq l(n')$. Wir werden nun den Satz beweisen, indem wir annehmen, dass $l(n, p) < l(n')$ und dass keine Funktionen mit Gödelnummer kleiner gleich n die A auswählt, quasikonstant ist. Dies bedeutet, dass K_A keine quasikonstante Funktion zur Berechnung von y aus x auswählen wird. Das wiederum heißt aber auch, dass es nur eine endliche Anzahl an Funktionen gibt, die nur eine endliche Anzahl an Programmen ausführen können. Da keines dieser Programme quasikonstant ist, bildet jedes auch nur eine endliche Anzahl von x Werten auf das gegebene y ab, womit insgesamt nur eine endliche Menge von (x, y) Tupeln existieren kann, was jedoch im Gegensatz zur Annahme der Flexibilität steht. □

Was diese gerade bewiesene Aussage so spannend macht, ist die Tatsache, dass es für jedes (x, y) Tupel quasikonstante Funktionen gibt (die x auf y abbilden) und dass diese auch in allen Funktionen mit endlicher Flexibilität Anwendung finden. Da in der Praxis diese Art von Computern eher die Regel als die Ausnahmen ist, ist es auch eher die Regel, dass die bedingte Beschreibungskomplexität auf Funktionen zurückgreift, die keine Aussage bezüglich der Komplexität eines Objekts erlauben. Besonders deutlich wird die Abhängigkeit der Komplexität eines Objekts von der zugrunde liegenden Nummerierung der Funktionenmenge. Tatsächlich hat diese größeren Einfluss als das Objekt selbst.

Neben der Abhängigkeit von der Nummerierung gibt es noch ein paar weitere Schwierigkeiten bei der Anwendung der Kolmogorovkomplexität auf Objekte. Besonders sei hier die Nichtberechenbarkeit zu nennen. Im Folgenden werden wir zeigen, dass $K(y) = K(y|\epsilon)$ nicht durch eine Turingmaschine berechenbar ist. Zuerst jedoch eine kurze Motivation: Die Nichtberechenbarkeit der Kolmogorovkomplexität ist sowohl eng verknüpft mit dem Halteproblem (Satz 2.4) als auch dem *Berry Paradoxon*. Dieses Paradoxon – zu dem es eine Vielzahl verschiedener Versionen gibt – steht repräsentativ für die Schwierigkeiten, die man bekommt, falls die Kolmogorov Komplexität berechenbar wäre. Es steht jedoch auch grundsätzlich für die Schwierigkeiten, die die Kolmogorovkomplexität mit sich bringt. Hier nun eine besonders prägnante Variante von Whitehead und Russel (1957, Kapitel 2, Seite 61):

> The number of syllables in the English names of finite integers tends to increase as the integers grow larger, and must gradually increase indefinitely, since only a finite number of names can be made with a given finite number of syllables. Hence the names of some integers must consist of at least nineteen syllables and among these there must be a least. ... But "the least integer not nameable in fewer than nineteen syllables„ is itself a name consisting of eighteen syllables; hence the least integer not nameable in fewer than nighteen syllables can be names in eighteen syllables which is a contradiction.

Besonders interessant wird dieses Paradoxon jedoch, wenn man sich mit der Bezeichnung, den „Namen", von natürlichen Zahlen beschäftigt. Whitehead und Russel gehen hierauf in einer Diskussion zu Berrys Paradoxon näher ein. Sie kommen zu dem Schluss, dass es eine totale Beschreibung (die im Sinne des Paradoxons alle möglichen Beschreibung beinhaltet) nicht geben kann. Eine Beschränkung auf Unterklassen von Beschreibungen würde das Paradoxon auflösen.

Im Fall der Kolmogorovkomplexität ist jedoch die Frage der Beschreibung klarer. Es geht um natürliche Zahlen, die als Folgen von Nullen und Einsen dargestellt werden. Für die Frage der Berechenbarkeit ist es notwendig zu wissen, ob eine Funktion tatsächlich für die Eingabe definiert ist oder nicht. Dies stellt eine Verbindung zum Halteproblem (Kleene 1971, Kapitel 13 und Rogers 1967, Kapitel 1.9) her, auf die wir nun näher eingehen werden. Zuerst ein notwendiger Hilfssatz auf dessen Beweis wir verzichten werden, er findet sich in Rogers (1967, Kapitel 1.8, Theorem V).

Hilfssatz 2.1. *Für alle $m, n \geq 1$ existiert eine rekursive Funktion s_n^m mit $m+1$ Variablen, sodass für alle $x, y_1, ...y_m$ gilt*

$$\lambda z_1 \cdots z_n[\varphi_x(y_1, ...y_m, z_1, ..., z_n)] = \varphi_{s_n^m(x,y_1,...,y_m)}(z_1, ..., z_n)$$

Dies ist die Church'sche λ-Notation (Rogers 1967, Introduction). Sie beschreibt auf knappe Weise die Definition einer rekursiven Funktion abhängig vom Term in den eckigen Klammern.

Satz 2.4 (Halteproblem – Rogers (1967))**.** *Es existiert keine rekursive Funktion g, sodass für alle x und y*

$$g(x,y) = \begin{cases} 1, & \textit{falls } \varphi_x(y) \textit{ konvergiert} \\ 0, & \textit{falls } \varphi_x(y) \textit{ divergiert} \end{cases}$$

gilt.

Beweis. Für diese Beweis definieren wir eine Funktion

$$\psi(z,x) = \begin{cases} 1, & \text{falls } \varphi_z(x,x) = 0 \\ \text{divergent}, & \text{falls } \varphi_z(x,x) \neq 0 \text{ oder divergiert.} \end{cases}$$

Diese ist nach Church's These partiell rekursiv und somit lässt sich nach Hilfssatz 2.1 eine Gödelnummer, in Abhängigkeit von z, finden. Anders formuliert: es gibt eine rekursive Funktion h, sodass für alle z $\varphi_{h(z)} = \lambda x[\psi(z,x)]$ gilt.

Gehen wir davon aus, dass für irgend ein z_0 $g = \varphi_{z_0}$ ist. Nach der Definition von h muss $\varphi_{h(z_0)}$ konvergent sein, genau dann, wenn $\varphi_{z_0}(x,x) = 0$, also wenn $\varphi_x(x)$ divergent (nach g) ist. Wenn wir nun x durch $h(z_0)$ ersetzen, erhalten wir

$$\varphi_{h(z_0)}(h(z_0)) \text{ konvergent} \Leftrightarrow \varphi_{z_0}(h(z_0), h(z_0)) = 0.$$

Aber wenn $\varphi_{z_0}(h(z_0), h(z_0)) = g(h(z_0), h(z_0)) = 0$ ist, so kann $\varphi_{h(z_0)}$ nicht konvergent sein. Also ist φ_{z_0} entweder undefiniert oder liefert falsche Informationen. □

Da sowohl das Halteproblem als auch die Kolmogorovkomplexität auf allen $x \in \mathbf{N}$ definiert sind, betrachten wir für die Berechenbarkeit auch nur totalrekursive Funktionen. Der Beweis der Nichtberechenbarkeit ist eine direkte Konsequenz des Halteproblems.

Satz 2.5 (Nichtberechenbarkeit von $K(x)$ – Cover und Thomas (2006))**.** *Die Funktion $K(x)$ ist nicht rekursiv.*

Beweis. Da es, nach Satz 2.4, keine partiell rekursive Funktion $g(x,y)$ gibt, die für alle Funktionen φ_x bestimmt, ob sie für die Eingabe y auch definiert sind, kann es auch keine partiell rekursive Funktion geben, die als charakteristische Funktion der Menge aller definierten Funktionen dient. Diese wäre jedoch notwendig, um die Menge aller Funktionen aufzustellen, deren Element mit minimaler Eingabelänge die Kolmogorovkomplexität definiert. □

Cover und Thomas (2006) begründet die Nichtberechenbarkeit mit der Laufzeit, die man bekommen würde, wenn ein Programm nicht terminiert. Es ist jedoch nicht vorher für jedes Programm klar, ob es terminiert oder nicht. Somit ist für K selbst nicht klar, ob es terminiert. Unabhängig von der Berechenbarkeit stellt sich noch die Frage der Approximation. Li und Vitànyi (1993, p. 103) haben in ihrem Buch einen Weg zur „Approximation“ der Kolmogorovkomplexität beschrieben. Wir werden auch im weiteren Verlauf von diesem Satz keinerlei Gebrauch machen. Er sei hier der Vollständigkeit halber jedoch erwähnt. Auf einen Beweis werden wir verzichten.
Satz 2.6 (Approximation of $C(x)$ – Li und Vitànyi (1993))**.** *Es existiert eine monoton mit t steigende absolut rekursive Funktion $\phi(t,x)$ für die gilt $\lim_{t\to\infty}\phi(t,x) = K(x)$.*

2.2 Informationsdistanzmaße

Bisher haben wir uns nur mit dem Informationsgehalt von Daten beschäftigt. In diesem Kapitel soll es nun darum gehen, wie man, aufgrund des Informationsgehalts die Ähnlichkeit zwischen Objekten bestimmen kann. Grundlage für die nun folgenden Überlegungen ist die bereits vorgestellte Kolmogorovkomplexität. Wir werden uns bei den Betrachtungen hauptsächlich an die Arbeiten von Bennett et al. und Cilibrasi und Vitànyi halten.

Distanzen finden in den verschiedensten Disziplinen der Mathematik Anwendung. Es gibt sie in unterschiedlichsten Ausprägungen. Allgemein folgen sie den Kriterien einer Metrik.
Definition 2.7 (Metrik)**.** *Sei Ω eine nichtleere Menge und $\mathbf{R}^+$ die Menge aller positiven reellen Zahlen. Eine Metrik, über Ω ist dann definiert als eine Funktion $d : \Omega \times \Omega \mapsto \mathbf{R}^+$ die folgende Kriterien erfüllt:*

- $d(x,y) = 0$ *g.d.w.* $x = y$*,*
- $d(x,y) = d(y,x)$ *und*
- $d(x,z) \leq d(x,y) + d(y,z)$*.*

Die Informationsdistanz (in engl. Information Distance), wie von Bennett et al. beschrieben definiert ein algorithmisches Ähnlichkeitsmaß.
Definition 2.8 (Informations- und Maximumsdistanz – Bennett et al. (1998))**.** *Die Informationsdistanz E zwischen x und y ist definiert durch*

$$E_A(x,y) = \min\{l(p) | A(p,x) = y,\ A(p,y) = x\}.$$

Die Maximumsdistanz ist gegeben durch

$$E_m(x,y) = \max\{K(y|x), K(x|y)\}.$$

Bennett et al. zeigen, dass E_A und E_m bis auf einen additiven logarithmischen Term äquivalent sind.

Die Informationsdistanz – die vom Konzept her wesentlich näher an der ursprünglichen Idee einer algorithmischen Distanz ist – zeigt ein ähnliches Verhalten wie die zugrundeliegenden Kolmogorovkomplexität. Mit der Wahl einer universellen rekursiven Funktion A handelt man sich ebenfalls einen Approximationsfehler ein. Dieser kann, wie beim Vorbild, beliebig groß werden und hängt nur von der Nummerierung der partiell rekursiven Funktionen ab. Es gilt also ebenfalls

$$E_A(x, y) \leq E_\varphi(x, y) + C_\varphi$$

mit den erwähnten Eigenschaften von C_φ. Die Distanz $E_A(x, y)$ – und noch wesentlich stärker E_m – zwischen zwei Objekten hängt also nur von der Nummerierung ab. Im konkreten Fall für x und y könnte die Nummerierung so gewählt werden, dass Funktionen, die das eine auf das andere Element abbilden, niedrige Gödelnummern haben. In so einem Fall wäre auch die Distanz zwischen den beiden gering. Auf einen Beweis verzichten wir, da dieser identisch ist zu den vorangegangenen.

In diesem Kontext ist es wichtig anzumerken, dass sich die Ergebnisse zur Verwendung konstanter Funktionen nur bedingt auf die Informationsdistanz übertragen lassen. Es lassen sich jedoch ebenfalls Fälle konstruieren, in denen einfache *Wenn-Dann*-Konstrukte Anwendung finden. Des Weiteren sollte, wie von Bennett et al. erwähnt, immer die Möglichkeit unendlich langer Berechnungen in Betracht gezogen werden, da die Ergebnisse sonst im klaren Widerspruch zu sogenannten Einwegfunktionen[1] stehen würden. Auch hier lässt sich anmerken, dass unendlich lange Zeichenketten in der praktischen Anwendung nur selten vorkommen.

Von praktischer Bedeutung ist noch ein weiterer Punkt. Die bisher vorgestellten Informationsmaße sind abhängig von der Länge der betrachteten Objekte. Hierzu beschränken wir uns auf Objekte $x, y \in \Omega_b = \{0, 1\}^*$ aus der Menge Ω_b der binären Zeichenketten.

Definition 2.9 (Normalisierte Distanz – Li et al. (2004))**.** *Eine normalisierte Distanz oder Ähnlichkeitsmetrik ist eine Funktion $d : \Omega_b \times \Omega_b \mapsto [0, 1]$, die symmetrisch ($d(x, y) = d(y, x)$) ist und für die für alle $x \in \Omega_b$ und alle Konstanten $e \in [0, 1]$*

$$|\{y \,|\, d(x, y) \leq e\}| < 2^{eK(x)+1}$$

gilt.

Anschaulich bedeutet dies, dass wir jedem Paar (x, y) einen Wert zwischen Null und Eins zuweisen können. Diesen Wert interpretieren wir als Abstand, wobei die Anzahl der Elemente, die einen bestimmten Abstand zueinander haben, abhängig von dem Abstand und der Kolmogorovkomplexität der Elemente ist.

Hier nun die Definition einer dazu passenden Distanz

[1] Einwegfunktionen sind Funktionen die leicht zu berechnen aber aufwendig umzukehren sind, hierzu gehören zum Beispiel Hash-Funktionen im Computersicherheitsbereich. Vgl. (Wikipedia 2010b)

Definition 2.10 (Normalisierte Informationsdistanz (NID) – Li et al. (2004))**.** *Die normalisierte Informationsdistanz (NID) ist, für zwei Elemente $x, y \in \Omega_b$ definiert als*

$$d_{\mathrm{NID}}(x,y) = \frac{\max\{K(x|y), K(y|x)\}}{\max\{K(x), K(y)\}}.$$

Zuletzt noch ein Satz zur NID, den wir jedoch nicht beweisen werden.

Satz 2.7. *Die normalisierte Informationsdistanz $d_{\mathrm{NID}}(x,y)$ minorisiert jede obere semiberechenbare normalisierte Distanz $f(x,y)$ mit $d_{\mathrm{NID}}(x,y) \leq f(x,y) + O(1/K)$ wobei $K = \min\{K(x), K(y)\}$.*

Die Schwierigkeit bei diesem Satz liegt hauptsächlich in der zugrunde liegenden Annahme, K lasse sich beliebig genau approximieren (Li et al. 2004, Definition II.3) und (Li und Vitànyi 1993, Theorem 2.7). Die Nähe der Kolmogorovkomplexität zum Halteproblem und die Tatsache, dass die Approximationsgüte des Halteproblems von der zugrundeliegenden Gödelnummerierung abhängt (Lynch 1974), legen nahe, dass solch allgemeine Aussagen, wie sie von Li und Vitànyi getroffen werden, fragwürdig[2] sind. Wir werden jedoch die bisher ausschließlich theoretischen Betrachtungen mit Leben füllen und zu handfesteren Überlegungen übergehen.

2.3 Kompressionsdistanzen

Kompression in ihren unterschiedlichen Formen beeinflusst schon seit Langem Mustererkennung und Statistik. Das zugrundeliegenden Konzept wird wohl am besten im sogenannten Minimum Description Length Principle (Grünwald 2004) beschrieben. Kurz und knapp lässt es sich mit den Worten „*die einfachste Lösung ist die Beste*“ zusammenfassen.

Cilibrasi und Vitànyi (2005) bauen auch auf dieser Überlegung auf. Sie stützen sich dabei auf die Kolmogorovkomplexität und die darauf basierende Informationsdistanz. Diese wollen sie mit Hilfe von Kompressionsalgorithmen approximieren. Ihr Ziel ist es, ein Verfahren zu entwickeln, das unabhängig von Merkmalen oder Parametern ist. Dieses parameterfreie Abstandsmaß (Keogh et al. 2004) soll dann jedes andere Abstandsmaß minorisieren. Konkret bedeutet dies, dass Cilibrasi und Vitànyi ein absolutes Maß entwickelt haben wollen, welches Abstände genauer darstellt als jedes andere Maß.

Für eine konkrete Formulierung dieses Maßes benötigen wir jedoch eine Definition der verwendeten Kompressoren und damit verbunden des Präfixcodes.

Definition 2.11 (Präfixcode – Bennett et al. (1998))**.** *Ein Präfixcode, oder auch präfixfreier Code ist eine Menge von Zeichenketten, wobei keine der enthaltenen Zeichenketten Präfix einer anderen ist.*

[2] Der Satz legt nahe, dass d_{NID} die Worte “Hund„ „Katze“ und „Auto“ einmal nach der Anzahl der Zeichen anordnet – damit wären sich “Hund„ und „Auto“ ähnlicher als “Hund„ und „Katze“ – gleichzeitig jedoch auch ein Maß bezüglich der biologischen Ähnlichkeit – “Hund„ und „Katze“ sind sich ähnlicher als “Hund„ und „Auto“ – minorisiert.

Präfixfreie Codes spielen bei der Definition verlustfreier Kompressoren eine zentrale Rolle.

Definition 2.12 (Verlustfreier Kompressor – Cilibrasi und Vitànyi (2005))**.** *Ein verlustfreier Kompressor $C_l : \Omega_b \mapsto \Omega_b$ weist jeder binären Zeichenkette einen Präfixcode zu, der eine exakte Rekonstruktion der binären Zeichenkette ermöglicht.*

Hier ist zu beachten, dass ein Kompressor nicht notwendigerweise Daten komprimiert. Wir werden in Abschnitt 4.6 noch eine weitere Definition von Kompressoren kennenlernen, deren Kernaspekt die Kompression ist. Zuerst werden wir uns mit den verlustfreien Kompressoren und deren Eigenschaften zur Mustererkennung beschäftigen.

Definition 2.13 (Normalisierte Kompressionsdistanz (NCD) – Cilibrasi und Vitànyi (2005))**.** *Sei $x, y \in \Omega_b$ und C ein Kompressor, dann ist die normalisierte Kompressionsdistanz von x und y wie folgt definiert*

$$d_{\mathrm{NCD}}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}.$$

Nun werden wir einige Sätze zur Normalisierten Kompressionsdistanz aufführen und beweisen. Das folgende Lemma stellt eine Verbindung zwischen einem normalisierten Distanzmaß und der Kraft-Ungleichung (Cover und Thomas 2006, Kapitel 5.2) her.

Lemma 2.1 (Cilibrasi und Vitànyi (2005))**.** *Ein Distanzmaß $d : \Omega \times \Omega \mapsto [0, 1]$, dass*

$$\sum_{y \neq x} 2^{-(1+d(x,y))C(x)} \leq 1$$

erfüllt, ist eine Normalisierte Distanz.

Beweis. Für den Beweis nehmen wir das Gegenteil an und zeigen, dass dies zu einem Widerspruch führt. Angenommen es existiert ein $e \in [0, 1]$, sodass die Bedingungen aus Definition 2.9 nicht erfüllt sind. Wir erhalten dadurch

$$\begin{aligned}
1 \geq & \sum_{y \neq x} 2^{-(1+d(x,y))C(x)} \\
\geq & \sum_{B} 2^{-(1+e)C(x)} \qquad \text{wobei} \qquad B = y \neq x \wedge d(x, y) \leq e \leq 1 \wedge C(y) \leq C(x) \\
\geq & 2^{-(1+e)C(x)+1} 2^{-(1+e)C(x)} > 1.
\end{aligned}$$

□

Deutlich besser greifbar als die Kraft-Ungleichung ist jedoch die Definition eines *normalen Kompressors*.

Definition 2.14 (Normaler Kompressor – Cilibrasi und Vitànyi (2005))**.** *Ein Kompressor $C : \Omega_b \mapsto \Omega_b$ heißt normal, falls er die folgenden (Un)Gleichungen erfüllt:*

1. *Idempotenz:* $C(xx) = C(x)$
2. *Monotonie:* $C(xy) \geq C(x)$
3. *Symmetrie:* $C(xy) = C(yx)$
4. *Distributivität:* $C(xy) + C(z) \leq C(xz) + C(yz)$.

Hierbei ist ein Fehler der Größe $O(\log n)$ *erlaubt, wobei* n *die maximale Länge eines Elements aus* Ω_b *ist.*

Mit Hilfe dieser Definition lässt sich die Normiertheit der NCD zeigen.

Satz 2.8. *Falls der in der NCD verwendete Kompressor normal ist, so ist die NCD ein normalisiertes Distanzmaß.*

Der Beweis (Cilibrasi und Vitànyi 2005) hierfür ist lang und trägt auch nicht unbedingt zum weiteren Verständnis der Thematik bei. Wir werden uns daher den praktischen Aspekten der NCD zuwenden. Thom hat hier eine sehr gut zu lesende Zusammenfassung präsentiert an der wir uns weitestgehend orientieren werden (Thom 2010).

2.3.1 Eigenschaften der NCD

Motiviert wurde die NCD durch die Ähnlichkeitsmetik [3] (Li et al. 2004). Sie ist mit der NCD, bis auf deren Verwendung von realen Kompressoren, weitgehend identisch. Aus diesem Grund werden auch in den Arbeiten von Li et al., Bennett et al., Cilibrasi und Vitànyi und Thom (2008) freizügig Eigenschaften der Ähnlichkeitsmetrik auf die NCD übertragen. Wir werden uns diese im Folgenden ansehen und beziehen uns dabei auf die zitierten Arbeiten.

Allgemeinheit - Eine besondere Eigenschaft der NCD ist die Parameterfreiheit nach Keogh et al. (2004). Dies bedeutet, dass es bei der NCD keine Parameter gibt, die an ein gegebenes Problem angepaßt werden müssen. Die NCD wählt automatisch – aufgrund der zugrundeliegenden Kolmogorovkomplexität – die passenden Aspekte aus, die für die Ähnlichkeitsberechnung notwendig sind.

Universalität - Da die Ähnlichkeitsmetrik – nach Satz 2.7 – jede andere Metrik minorisiert, gehen einige der eingangs erwähnten Autoren davon aus, dass auch die NCD dazu in der Lage ist. Dies würde bedeuten, dass für ein gegebenes Objekt jede mögliche Ähnlichkeitsbetrachtung durch die NCD vorgenommen wird. Cilibrasi und Vitànyi sind hier etwas vorsichtiger und beschränken sich auf den später noch aufgeführten Satz 2.9. Dieser besagt, dass die NCD für sehr spezielle Objekte jede andere Metrik minorisiert.

[3] Li et al. sind der Meinung, dass die Tatsache, dass die von ihnen vorgestellte Ähnlichkeitsmetrik alle anderen Metriken (zumindest nach ihrer Vorstellung) minorisiert, ausreicht, um sie DIE Ähnlichkeitsmetrik zu nennen. Da sich für alle anderen gängigen Maße eigenständige Namen entwickelt haben, sollte dies nicht zu Mißverständnissen führen.

Robustheit - Die NCD ist in sofern robust, als dass sie unabhängig von der Wahl der zugrundeliegenden Kompressionsverfahren ist. Dies wurde in zahlreichen Arbeiten (Burkovski 2009; Thom 2008; Keogh et al. 2004; Cilibrasi und Vitànyi 2005) wiederlegt, indem festgestellt wurde, dass einige Kompressoren für bestimmte Probleme besser geeignet sind als andere.

Wie schon bei dem Punkt Universalität angedeutet, ist die generelle Minorisierung jeder anderen Metrik durch die NCD nicht haltbar. Es gibt jedoch eine abgeschwächte Variante, die von Cilibrasi und Vitànyi (2005) mit folgendem Satz vorgestellt wird.

Satz 2.9. *Für eine berechenbare Ähnlichkeitsmetrik d, eine Konstante $k \geq 0$ und $x, y \in \Omega_b$ wobei $C(xy) - K(xy) \leq a$ ist gilt $d_{\mathrm{NCD}}(x, y) \leq d(x, y) + (a + O(1))/k$ mit $k = \max\{C(x), C(y)\}$.*

Wir verzichten auf einen Beweis und werden auch nicht näher auf Aspekte wie die Größe von a oder der Konstante $O(1)$, eingehen.

2.4 Zusammenfassung

Die in diesem Kapitel vorgestellten Verfahren und Methoden haben zur Berechnung von Ähnlichkeiten einen sehr zwiespältigen Eindruck hinterlassen. Die Kolmogorovkomplexität als absolutes Informationsmaß besticht durch ihre Klarheit und Einfachheit. Jedoch ist sie, wie wir gezeigt haben, nur bedingt universell einsetzbar. Ihre Aussagekraft ist für viele Elemente eben nicht intuitiv einleuchtend. Gerade was die Approximation der Kolmogorovkomplexität angeht, so gibt es begründete Zweifel an den Aussagen von Li und Vitànyi. Diese fundamentale Eigenschaft ist jedoch Basis für die meisten weiteren Aussagen bezüglich der von Bennett et al. vorgestellten Ähnlichkeitsmetrik. Die meisten in diesem Kapitel vorgestellten Arbeiten beziehen sich in der einen oder anderen Form auf die Universalität der NID oder der NCD. All die dort getroffenen theoretischen Aussagen – aber auch die darauf basierenden Erklärungsversuche für experimentelle Ergebnisse – stehen somit ebenfalls auf wackligen Beinen. Nichtsdestotrotz sind die experimentell in den Arbeiten gewonnenen Erkenntnisse valide. Sie dokumentieren eine erstaunliche Leistungsfähigkeit von kompressionsbasierten Verfahren. Im folgenden Kapitel werden wir tiefere Einblicke in die Funktionsweise dieser Verfahren gewinnen und mögliche Anwendungsszenarien vorstellen.

KAPITEL

3 PRAXIS DER NCD

Überblick

- Für die NCD sind die Kompressionsverfahren von Huffman (Absch. 3.1.1), Ziv und Lempel (Absch. 3.1.3) am bedeutendsten.
- Die NCD eignet sich zur Identifizierung identischer Entitäten (Absch. 3.2.1). Dabei ist besonders die Robustheit gegenüber Rauschen ein Alleinstellungsmerkmal.
- Das Hauptmerkmal der NCD sind zusammenhängende Zeichenketten (Absch. 3.2.2). Je länger diese sind, umso mehr besteht die Chance, dass eine Semantik zugeordnet werden kann.
- Auf kontinuierlichen Daten werden nur sehr kurze Zeichenketten erkannt. Die NCD beschränkt sich daher bei Bildern auf einen Vergleich der häufig vorkommenden Farben (Absch. 3.2.2).

Die Normalisierte Kompressionsdistanz (Normalized Compression Distance – NCD) hat in den letzten Jahren großen Anklang in der Fachwelt gefunden. Dies ist zum Einen auf das theoretische Fundament, das wir im vorangegangenen Kapitel besprochen haben, aber auch auf praktische Erfolge zurückzuführen. In den zentralen Arbeiten von Cilibrasi und Vitànyi sowie Li et al., aber auch in eher der Anwendung näherstehenden Arbeiten, wie der von Heidemann und Ritter oder der des Autors (Klenk et al. 2009b) werden Beispiele für die Fähigkeiten der NCD gegeben.

In diesem Kapitel werden wir uns der anwendungsbezogenen Komponente der NCD widmen. Dies kann aufgrund der großen Diskrepanz zwischen der nichtberechenbaren Kolmogorovkomplexität (und damit verbunden auch der NID) und den

tatsächlich berechenbaren Kompressoren völlig losgelöst von der theoretischen Motivation geschehen. Gerade dieser große Abstand zwischen Theorie und Praxis wird es uns in einem späteren Kapitel erlauben, eine alternative Theorie zu präsentieren. Wir werden daher die hier vorgestellten Experimente nicht in Relation zu der bereits entwickelten Motivation setzen, sondern sie als Indiz dafür werten, dass Kompression in der Mustererkennung eine große Rolle spielt und die Suche nach einer allgemeinen Theorie vielversprechend ist. Wie das theoretische Fundament dazu aussieht, und warum dieses besser in der Lage ist die Ergebnisse zu erklären, werden wir in den Kapiteln 4 und folgend darstellen.

3.1 Kompressionsverfahren

Wie der Name Normalisierte Kompressionsdistanz schon sagt, spielen Kompressoren eine wesentliche Rolle in diesen Verfahren. Es geht nicht nur um ein einziges Verfahren, die NCD steht vielmehr für eine ganze Reihe von verschiedenen Mustererkennungsverfahren, wobei die zentrale Herangehensweise immer gleich ist. Der Unterschied liegt im verwendeten Kompressor. Dieser repräsentiert in gewisser Weise die eigentliche Mustererkennung, denn er wählt die Merkmale aus, die zur Erkennung herangezogen werden. Um dies zu verstehen, ist es unerlässlich, die genaue Funktionsweise dieser Verfahren zu kennen. Wir werden uns deshalb in diesem Abschnitt die bekanntesten Kompressionsalgorithmen genauer ansehen. Da wir uns ausschließlich um verlustfreie Verfahren kümmern, beschränken wir uns auf gängige Algorithmen wie zum Beispiel Huffmancodierung oder die Verfahren von Ziv und Lempel (1977, 1978) sowie *bzip2* von Julian Seward (Wikipedia 2010a).

3.1.1 Huffmancodierung

Geht man davon aus, dass jedes Symbol[1] in einem Dokument – wir gehen hier von diskreten, also auf eine abzählbaren Menge von Basissymbolen basierenden Daten aus – mit gleicher Wahrscheinlichkeit auftritt, so ist es berechtigt, dass jedes Symbol die gleiche Länge[2] hat. Falls jedoch einzelne Symbole öfter verwendet werden als andere, ist es hilfreich, diese kürzer zu codieren. Huffman nutzte diese Überlegung für sein bekanntes Kompressionsverfahren, das wir hier näher betrachten werden. Gegeben sei eine endliche Menge von Symbolen $\mathcal{S} = \{s_1, ..., s_n\}$. Zu jedem Element s_i kennen wir die Wahrscheinlichkeit $P(s_i)$, mit der es in einem Dokument auftritt, und dessen

[1] Wir werden im Folgenden die Begriffe Symbol und Zeichen synonym verwenden. Aus dem Zusammenhang sollte jeweils klar werden, worum es sich genau handelt, ein abstraktes Symbol oder ein konkretes Zeichen.

[2] Die Symbole, von denen wir hier sprechen, sind binär codiert und können somit auch eine Länge größer eins haben.

HUFFMAN (D)
  $n \leftarrow |D|$
  $Q \leftarrow D$
  for $i = 1$ to $n - 1$ **do**
    neuen Knoten z bereitstellen
    $left[z] \leftarrow x \leftarrow$ EXTRACT-MIN(Q)
    $right[z] \leftarrow y \leftarrow$ EXTRACT-MIN(Q)
    $p[z] \leftarrow p[x] + p[y]$
    INSERT(Q,z)
  end for
  return EXTRACT-MIN(Q)

Algorithmus 1: Huffmancodierung nach (Cormen et al. 2001). Die Eingabevariable D entspricht dem Dokument und Q der Menge der Symbole. Die Funktion EXTRACT-MIN(Q) wählt das Element x mit der geringsten Wahrlscheinlichkeit $p(x)$ aus.

Länge $L(s_i)$. Die durchschnittliche Länge eines zufällig aus einem Dokument gewählten Symbols ist daher

$$L_D = \sum_{i=1}^{n} L(s_i) \cdot P(s_i) \qquad \text{wobei} \qquad \sum_{i=1}^{n} P(s_i) = 1.$$

Wir suchen nach einer Codierung für Dokumente, für die L_D minimal wird. Zusätzlich müssen die folgenden zwei Beschränkungen erfüllt werden:

1. Keine zwei Symbole werden mit identischen Abfolgen von Binärzeichen codiert.

2. Kennt man den Startpunkt einer Sequenz, so sind keine weiteren Informationen bezüglich Anfang oder Ende der einzelnen Symbole notwendig.

Die zweite Einschränkung erfordert, dass kein Symbol Präfix eines anderen Symbols ist. Man beachte, dass es in allgemein üblichen Binärcodierungen kein Trennzeichen zwischen den Symbolen gibt. Mit Hilfe dieser Einschränkungen lässt sich ein Code generieren, der eindeutig decodierbar und auch optimal ist. Der Code selbst weist den Symbolen Binärsequenzen zu, die abhängig von ihrer Auftrittswahrscheinlichkeit unterschiedliche Längen haben. Die Idee dabei ist es, dass die zwei unwahrscheinlichsten Symbole die selbe Länge haben und sich nur im letzten Bit unterscheiden. Dies führt zu dem in Algorithmus 1 beschriebenen „Greedy“ Verfahren. In diesem Algorithmus werden die zwei Symbole mit der geringsten Wahrscheinlichkeit zu einem neuen Symbol mit kombinierter Wahrscheinlichkeit verschmolzen. Dieses neue Symbol wird wieder – entsprechend seiner Wahrscheinlichkeit – in die Liste eingefügt. Die letzten zwei Symbole bekommen immer 0 (für das nun vorletzte), oder 1 (für das letzte) zur

aktuellen Codierung hinzugefügt. Handelt es sich um ein verschmolzenes Symbol, bekommen alle Ursprungssymbole die Zahl hinzugefügt.

Zum besseren Verständnis werden wir uns nun ein Beispiel aus dem Buch von Cover und Thomas (2006, Beispiel 5.6.1) ansehen.

Beispiel 3.1. *Wir haben eine Zufallsvariable X gegeben, die Werte aus der Menge $\mathcal{X} = \{s_1, s_2, s_3, s_4, s_5\}$ annehmen kann. Die Wahrscheinlichkeiten für das Eintreten der jeweiligen Werte ist $P([s_1, s_2, s_3, s_4, s_5]) = [0.25, 0.25, 0.2, 0.15, 0.15]$. Entsprechend des Huffman-Algorithmus wählen wir nun die zwei Werte mit der niedrigsten Wahrscheinlichkeit, also 4 und 5. Diese sollen sich in der Codierung nur im letzten Bit unterscheiden, also weisen wir dem Letzten (von bislang noch unbekannt vielen Bits) einmal die 0 und einmal die 1 zu, $C(4) = *0$ und $C(5) = *1$. Aus diesen zwei Symbolen generieren wir ein neues Symbol s_{45} mit der kombinierten Wahrscheinlichkeit $P(s_{45}) = 0.15 + 0.15 = 0.3$. Dieses neue Symbol sortieren wir in die Liste ein und entfernen nun die nächsten zwei Symbole mit geringster Wahrscheinlichkeit (2,3). Diesen weisen wir nun an der vorletzten Stelle 0 und 1 zu. Wir generieren wieder ein neues Symbol s_{23} mit Wahrscheinlichkeit 0.45 und fügen es am Anfang der Liste ein. Diese hat jetzt die Form $[s_{23}, s_{45}, s_1]$ mit den Wahrscheinlichkeiten $[0.45, 0.3, 0.25]$. Nun wählen wir uns wieder die letzten beiden Symbole und weisen ihnen an der letzten noch nicht zugewiesenen Stelle 0 und 1 zu. Da es sich bei s_{45} um ein kombiniertes Symbol handelt, weisen wir s_4 und s_5 eine 0 zu und s_1 eine 1. Dieses Prozedere wiederholen wir so lange, bis es nur noch ein Gesamtsymbol mit Wahrscheinlichkeit 1 gibt. Übersichtlich wird das Ganze auch in Tabelle 3.1 dargestellt.*

Die Funktionsweise des Verfahrens ist verblüffend einfach. Um so erstaunlicher ist die Tatsache, dass es einen optimalen Code liefert. Konkret heißt das, dass kein anderer, eindeutig decodierbarer Code besser komprimiert als ein Huffmancode. Cover und Thomas zeigen dies mit dem folgenden Satz, auf dessen Beweis, der sich in Cover und Thomas (2006, Theorem 5.8.1) findet, wir verzichten werden.

Satz 3.1 (Optimalität des Huffmancodes)**.** *Ein Huffmancode ist optimal; falls C^* ein Huffmancode ist und C' ein anderer eindeutig decodierbarer Code ist, so gilt $L(C*) \leq L(C')$.*

Mit Hilfe der Huffmancodierung können wir Daten, deren Verteilung uns bekannt ist, sehr einfach und effektiv komprimieren. Sie nutzt jedoch nicht die Eigenschaft aus, dass das Auftreten von Symbolen korreliert sein könnte. Gerade bei Texten zum Bei-

Code	X	Wahrscheinlichkeit				
01	s_1	0.25	0.3	0.45	0.55	1
10	s_2	0.25	0.25	0.3	0.45	
11	s_3	0.2	0.25	0.15		
000	s_4	0.15	0.2			
001	s_5	0.15				

Tabelle 3.1: Huffmancode für Beispiel 3.1

spiel treten Symbole gehäuft auf, auf ein un folgt in sehr vielen Fällen ein d. Dies ist jedoch nicht nur auf Texte beschränkt auch von Maschinen generierten Daten weisen stark lokale Beziehungen zwischen den Zeichen auf.

3.1.2 Lauflängencodierung

Ein sehr deutlicher Hinweis darauf, dass das Auftreten von Symbolen korreliert sein könnte ist, dass identische Symbole immer gehäuft vorkommen. Solche zusammenhängenden, einheitlichen Bereiche werden von der Lauflängencodierung erkannt und komprimiert. Dabei wird ein Symbol, das mehr als einmal hintereinander vorkommt durch das Symbol selbst und die Anzahl, wie oft es wiederholt wird, ersetzt. Man erhält auf diese Weise eine Sequenz von Symbol-Anzahl Tupeln. Ersetzt man nur die Symbolsequenzen für die es sich lohnt (z.B. erst falls drei und mehr identische Symbole auftreten), durch solch ein Tupel, benötigt man ein Trennzeichen, um ein Tupel von zwei regulären Symbolen zu unterscheiden. Hier ein kurzes Beispiel (aus Burkovski 2009)

Beispiel 3.2. *Betrachtet man die folgende Zeichenkette*

$$w = aaabbbbbccdaaaaaaaaaaa \qquad \textit{mit Alphabet} \qquad \mathcal{A} = \{a, b, c, d\}$$

so kann man leicht sehen, dass es hier ein großes Potential für Kompression gibt. Nehmen wir an, wir codieren jedes Zeichen mit zwei Bit. Das würde zu einer 44 *Bit langen binären Repräsentation führen. Nutzen wir jedoch die Lauflängencodierung so codieren wir weiterhin jedes Zeichen mit zwei Bit, wir codieren jedoch auch noch die Anzahl der Wiederholungen mit 3 Bit. Zusammen erhalten wir für jedes Zeichen samt Wiederholung 5 Bit und die folgende Zeichenkette*

$$w_c = 3a5b2c1d8a3a$$

was einer Länge von 30 *Bit entspricht. Da wir eine feste Länge für die Zeichen-Anzahl Tupel angenommen haben, muss hier auch kein Präfixcode für eine eindeutige Codierung verwendet werden.*

3.1.3 Lempel-Ziv Verfahren

Neben der Wiederholung identischer Symbole gibt es noch andere Formen der Korrelation. In Textdaten führen Zusammenhänge zwischen Buchstaben, wie im Deutschen zwischen c und h was sehr oft als ch auftritt, zu einer starken Korrelation zwischen den einzelnen Symbolen. Ein weiteres Beispiel hierfür wären Folgen wie und oder allgemeine Worte. Dies lässt sich jedoch nicht mehr mit der Lauflängencodierung abbilden und wir benötigen ein mächtigeres Verfahren.

Die grundlegende Idee zu den Verfahren von Lempel und Ziv entstammt der Untersuchung der Komplexität, oder – gleichbedeutend in diesem Kontext – der Zufälligkeit, von Zeichenketten. In ihrer Arbeit aus dem Jahr 1976 (Lempel und Ziv 1976)

untersuchten sie, wie man mit Hilfe von Wörterbüchern und Referenzen auf diese Wörterbücher quasizufällige Zeichenketten generieren oder die Komplexität der Zeichenkette bestimmen kann. Versucht man nämlich, jegliche Korrelation (zwischen dem Auftreten einzelner Symbole), die in Zeichenfolgen enthalten ist, zu reduzieren, indem man wiederkehrende Muster durch neue, einzelne Symbole ersetzt, so kommt man den gewünschten quasizufälligen Folgen schon sehr nahe. Diese Vorgehensweise wird detailliert in der Arbeit von 1976 (Lempel und Ziv 1976) beschrieben. In den darauf folgenden Jahren entwickelten sie, aufbauend auf der Idee zufälliger Zeichenfolgen, die bekannten Kompressionsverfahren (Ziv und Lempel 1977, LZ77 und Ziv und Lempel 1978, LZ78). Der große Durchbruch kam unter anderem im Jahr 1984 mit dem Verfahren von Welch (1984), das unter dem Kürzel LZW bekannt geworden ist.

Sliding Window

Der Sliding Window Ansatz nutzt ein abstraktes Wörterbuch, bestehend aus zwei Puffern, dem Suchfenster und dem Vorschaufenster. Das Verfahren sucht dabei nach der längsten Zeichenkette im Suchfenster, die mit der aktuellen Zeichenkette im Vorschaufenster übereinstimmt. Wird eine solche gefunden, wird sie durch ein Tupel, bestehend aus der Referenz (einen Zeiger auf den Beginn der Zeichenkette im Suchfenster), deren Länge und dem ersten Zeichen im Vorschaufenster, das nicht mehr zur gefundenen Zeichenkette gehört, ersetzt. Wird keine Übereinstimmung gefunden, werden Referenz und Länge auf Null gesetzt und mit dem nächste Zeichen im Tupel gespeichert.

Der Algorithmus beginnt mit einem Such- und Vorschaufenster fester Größe, wobei das Suchfenster deutlich größer sein sollte, als das Vorschaufenster. Gängige Werte für das Vorschaufenster sind 256 Byte und 32 Kilobyte für das Suchfenster. Ausgehend von der aktuellen Position, wird der Suchpuffer nach dem aktuellen Zeichen durchsucht. Wird es gefunden, wird überprüft, wie viele Zeichen sich ausgehend von dem aktuellen entsprechen. Dies wird für alle Zeichen wiederholt und die längste identische Zeichenkette wird ausgewählt. Nun wird die aktuelle Zeichenkette (und das nachfolgende Zeichen) durch das Tupel $\langle p, l, x\rangle$ ersetzt, wobei p die Position (ausgehend von der aktuellen Position) im Suchfenster, l die Länge der gefundenen Zeichenkette und x das erste nicht mehr in der Zeichenkette vorkommende Zeichen ist. Such- und Vorschaufenster werden nun um die Länge der gefundenen Zeichenkette verschoben, sodass die aktuelle Position das erste Zeichen nach x ist. Im Folgenden werden wir uns anhand eines Beispiels (aus Burkovski 2009) die Funktionsweise von LZ77 vor Augen führen.

Beispiel 3.3. *Ausgehend von der Zeichenkette*

$$w = abbcccbcabbcacabbc$$

betrachten wir das Vorgehen von LZ77, wobei wir eine Suchfensterlänge von 7 Zeichen und eine Vorschaufensterlänge von 5 Zeichen annehmen. Nun beginnen wir mit der Codierung. Tabelle 3.2 zeigt die einzelnen Schritte. In den Schritten eins und zwei wird jeweils kein passendes Symbol gefunden, in Schritt drei hingegen das Zeichen 'b' was

Schritt	Suchfenster							Vorschaufenster					Tupel
1								a	b	b	c	c	(0,0,a)
2							a	b	b	c	c	c	(0,0,b)
3						a	b	b	c	c	c	b	(1,1,c)
4				a	b	b	c	c	c	b	c	a	(1,1,c)
5		a	b	b	c	c	c	b	c	a	b	b	(4,2,a)
6	b	c	c	c	b	c	a	b	b	c	a	c	(7,1,b)
7	c	c	b	c	a	b	b	c	a	c	a	b	(4,2,c)
8	c	a	b	b	c	a	c	a	b	b	c		(6,4,EOF)

Tabelle 3.2: LZ77 Verfahren für Beispiel 3.3

mit dem aktuellen Zeichen korrespondiert. Dieses kann nun durch ein Tupel ersetzt werden. Ebenso verläuft es in Schritt vier. In Schritt fünf werden zum ersten Mal mehr Zeichen gefunden. Dort kann man auch gut erkennen, dass die Position (4*) von der aktuellen Position aus bestimmt wird und nicht vom Anfang des Fensters. Die Länge beträgt* 2 *und das nachfolgende Zeichen ist 'a'. Der restliche Verlauf entspricht einer Wiederholung des gerade beschriebenen Vorgehens.*

Tree-Structured

Im Gegensatz zum gerade beschriebenen pufferbasierten Ansatz des LZ77, wird beim LZ78 eine Baumstruktur aufgebaut. Sie zielt darauf ab, dass nur Einträge in das Wörterbuch aufgenommen werden, die minimal kurz sind und nicht bereits im Wörterbuch existieren. Das geschieht indem man den längsten, mit der aktuellen Zeichenkette identischen, Wörterbucheintrag um das nächste Zeichen ergänzt. Auf diese Weise erhält man einen neuen und längeren Wörterbucheintrag, durch den man die aktuelle Zeichenkette ersetzen kann. Die Folge $w = abbcccbcabbcacabbc$ aus dem obigen Beispiel führt zu den Wörterbucheinträgen $\mathcal{W} = \{a, b, bc, c, cc, bca, ...\}$ und damit zu den komprimierten Repräsentation $w_c = (0, a)(0, b)(2, c)(0, c)(4, c)(3, a)...$. Eine Optimierung dieser Vorgehensweise ist das LZW Verfahren (Welch 1984), das in Algorithmus 2 aufgeführt ist.

Nachdem wir uns nun einzelne Kompressionsverfahren näher angeschaut haben, wird es Zeit, sich mit deren Tauglichkeit für die Mustererkennung zu beschäftigen. Im nun folgenden Abschnitt werden wir uns Beispiele ansehen und Experimente zur NCD und deren Funktionsweise durchführen.

```
LZW (input)
  dict ← INITDICT()
  w ← ϵ
  for all c in input do
    if w + c ∈ dict then
      w ← w + c
    else
      OUTPUT(dict.index[w])
      dict ← ADDENTRY(w + c)
      w ← c
    end if
  end for
  OUTPUT(dict.index[w])
```

Algorithmus 2: Der LZW Algorithmus. Die Funktion INITDICT() erstellt eine Wörterbuch das nur aus den Basissymbolen besteht, ADDENTRY(x) fügt das Wort x zum Wörterbuch hinzu und OUTPUT(x) schreibt x in die Ausgabe.

3.2 NCD auf diskreten Daten

Diskrete Daten bildeten von Anfang an den zentralen Anwendungsbereich der Normalisierten Kompressionsdistanz (NCD). Dies liegt zu einem großen Teil an der inhärent diskreten Struktur der Daten. Damit sind nicht nur verlustfreie Kompressionsverfahren, die auf diskreten Symbolen arbeiten gemeint, sondern auch die Kolmogorovkomplexität selbst, die auf kontinuierlichen Daten nicht definiert ist.

Wir werden uns in diesem Abschnitt intensiv mit der Funktionsweise der NCD für die Mustererkennung beschäftigen. Wie die Kompressionsverfahren arbeiten, haben wir im vorangegangenen Abschnitt gesehen. Wie sich dies auf die Mustererkennung auswirkt, ist auf den ersten Blick nicht zu erkennen. Wir werden uns daher Experimente ansehen, die Aufschluss geben werden über zentrale Fragen der Mustererkennung. Unser Hauptaugenmerk wird dabei auf den betrachteten Merkmalen liegen.

Bevor wir jedoch allzu tief in die Funktionsweise der NCD eintauchen, werden wir uns ein Anwendungsbeispiel ansehen, das demonstrieren wird, wie die NCD angewendet wird. Weitere Details finden sich in der Arbeit des Autors (Klenk et al. 2009b).

3.2.1 Anwendungsfall: Entity Identification

Die Integration verschiedener Datenquellen stellt eine immer größere Herausforderung im Rahmen der Wissensextraktion aus Datenbanken dar. Datenquellen in Unternehmen zum Beispiel werden immer spezifischer und auf die einzelnen Abteilungen ausgerichtet, sodass Informationen über mehrere Datenbanken in unterschiedlichen Formaten verteilt sind. Gleichzeitig weckt das kontinuierliche Wachstum an Rechenleistung Begehrlichkeiten; eine immer allumfassendere Sicht auf die Daten und das

Unternehmen wird gefordert. Stark verteilte Datenquellen und der Wunsch nach einer holistischen Sicht verstärken den Bedarf an einer integrierten Datenhaltung.

Im Rahmen dieses Beispiels werden wir uns mit dem Thema Entity Identification beschäftigen. Konkret bedeutet dies, Einheiten (Entities) über mehrere heterogene Datenquellen hinweg zu identifizieren. Worin genau die Schwierigkeiten dabei liegen und wie uns die NCD helfen kann, werden wir im Folgenden sehen.

Gerade im medizinischen Bereich ist die Integration unterschiedlichster Datenquellen von besonderer Bedeutung. Patientendaten sind üblicherweise über viele Abteilungen verstreut. So liegen zum Beispiel die allgemeinen Stammdaten in einem Klinik-Informationssystem, Labordaten werden in einem Labor-Managementsystem gespeichert, und Fachabteilungen wie die Radiologie haben wieder eigene Systeme. Für eine holistische Betrachtung ist es jedoch von zentraler Bedeutung, diese Daten zusammenzuführen. Das heißt, man muss die relevanten Informationen aus den einzelnen Quellen extrahieren und die zu den Datensätzen gehörenden Patienten identifizieren. Diese Aufgabe ist einfach, wenn man gemeinsame, im besten Fall eindeutige, Patientendaten in allen Quellen findet. Hat man diese nicht zur Hand, muss man versuchen, über Umwege den Patienten zu erkennen. Hier kann die Kombination aus Namen und Geburtsdatum oder Namen und Adresse gute Dienste leisten. Das Problem dabei ist jedoch, dass diese Daten oft nicht konsistent geführt werden. Im einen Fall ist noch der Mädchenname gespeichert oder beim Geburtsdatum ist nur das Jahr bekannt, im anderen Fall jedoch das volle Geburtsdatum und der Ehename. Was wir also benötigen, ist eine Methode, die möglichst sicher auf identische Patienten schließt, auch wenn die Daten unvollständig und „verrauscht“ sind. Darüber hinaus sollte diese Methode noch möglichst schnell zu berechnen sein, da sehr viele Daten verglichen werden müssen.

Wir werde uns im Folgenden die Entity Identification mit Hilfe der Normalisierten Kompressionsdistanz ansehen. Wie wir sehen werden, hat deren Einsatz, besonders was Geschwindigkeit und Robustheit angeht, einige Vorteile gegenüber anderen gängigen Methoden. Doch zuerst werden wir einen Blick auf Entity Identification im Generellen werfen.

Entity-Erkennung

Die Standardvorgehensweise zur Erkennung identischer Einheiten in unterschiedlichen Datenquellen ist die Folgende: Zuerst werden gemeinsame Datenfelder identifiziert, was üblicherweise manuell geschieht. Dann berechnet man Abstände oder Ähnlichkeiten zwischen den Werten der einzelnen Felder und vereinigt diese zu einem generellen Abstandsmaß. Dieses gibt an, wie ähnlich sich die beiden Datensätze sind und damit auch, wie wahrscheinlich es ist, dass sie zu ein und der selben Entität gehören. Dieser Prozess ist sehr fehleranfällig. So können zum Beispiel feldübergreifende Fehler (Vor- und Nachname sind vertauscht) nicht erkannt oder strukturelle Unterschiede der Datenquellen (in einem Datensatz gibt es nur ein Adressfeld, der andere Datensatz hat jedoch unterschiedliche Felder für Straße, Ort und Postleitzahl) nicht behandelt

werden. Formal betrachtet kann man diese Probleme in drei Kategorien unterteilen: *Lexikale Heterogenität*, *strukturelle Heterogenität* und das Fehlen von *Datenqualität*.

Aktuelle Forschungsarbeiten betrachten die gerade angesprochenen Probleme üblicherweise unabhängig voneinander. Zuerst wird die Struktur der Datensätze vereinheitlicht, dann werden identische Felder einander zugeordnet. Bereits im Jahre 1969 haben Fellegi und Sunter die Notwendigkeit beschrieben, korrespondierende „Blöcke“ zu identifizieren und für eine Weiterverarbeitung zugänglich zu machen (Fellegi und Sunter 1969). Einen ähnlichen Ansatz beschreibt der von Winkler (2006) gegebene Überblick. Andere Arbeiten, wie zum Beispiel die von Elmagarmid et al. (2007) oder Hernández und Stolfo (1998), konzentrieren sich ausschließlich auf die lexikale Heterogenität.

In ihrem Standardwerk zum Thema Data Warehouses erklären Kimball und Caserta (2004), dass 70% des Entwicklungsaufwandes für ein solches Datenwarenhaus der Extract-Transform-Load (ETL) Prozess verschlingt. Ein Großteil dieser Arbeit bezieht sich auf die manuelle Identifikation einzelner Entitäten. Im Gegensatz hierzu benötigt der im Folgenden beschriebene Ansatz keinerlei Handarbeit. Mit dem Ergebnis, dass die aufwändige Identifikation zusammengehöriger Felder entfällt.

Der eigentliche Abgleich zusammengehöriger Felder ist üblicherweise entweder *Lern-* (Sarawagi und Bhamidipaty 2002; Zhao und Ram 2005; Zhao 2007; Zhao und Ram 2008; Bilenko und Mooney 2003) oder *Regelbasiert* (Hernández und Stolfo 1998; Elmagarmid et al. 2007; Navarro 2001). Jedoch haben beide Vorgehensweisen Nachteile. Der lernbasierte Ansatz erfordert Trainingsdaten, die in der Regel aufwändig von Hand erstellt werden müssen. Hierbei ist es wichtig, dass man so viele *true-positive* Fälle wie möglich abdeckt. Dies bedeutet, dass man den Klassifikator möglichst unabhängig von Einzelbeispielen hält. Eine besondere Herausforderung stellen jedoch nicht die positiven Beispiele, sondern realitätsnahe Negativbeispiele dar. Sarawagi und Bhamidipaty (2002) umschiffen dieses Problem mit Hilfe von *active learning*. Dabei werden aktiv neue Trainingsbeispiele generiert und dem Benutzer präsentiert.

Eine Besonderheit stellen sogenannte distanzbasierte Verfahren dar. Sie benötigen üblicherweise weder Trainingsdaten noch eine vorgegebene Regelmenge. Was bei ihnen jedoch kritisch ist, ist die Wahl eines geeigneten Grenzwerts, ab dem Beispiele als negativ verworfen werden; diesen Aspekt werden wir jedoch später noch genauer untersuchen.

Regelbasierte Verfahren benötigen ein solides Fundament an Hintergrundwissen. Die Daten sollten nicht zu viel Rauschen enthalten und die Struktur sollte einheitlich sein (Elmagarmid et al. 2007).

Wir werden uns nun im Folgenden ein Verfahren ansehen, dass auf der Normalisierten Kompressionsdistanz basiert. Wie wir sehen werden, kommen zahlreiche, bereits angedeutete Eigenschaften der NCD, wie zum Beispiel deren Robustheit gegenüber Rauschen (Cebrian et al. 2007), der Verzicht auf Trainingsdaten und noch einige weitere (Heidemann und Ritter 2008; Cilibrasi und Vitànyi 2005; Amitay et al. 2007) zum Einsatz.

ETL und Dublettenprüfung

Trotz ihres hohen Alters ist die Entitätenerkennung, besonders im Rahmen der Datenintegration und dem ETL Prozesses, ein aktives Forschungsfeld. Dies wird besonders deutlich, wenn man sich die zahlreichen aktuellen Publikationen (Zhao 2007; Zhao und Ram 2008; Goiser und Christen 2006; Christen 2007; Yan et al. 2007, um nur einige zu nennen) ansieht. Neben einer aktiven Forschergemeinde hat das Thema aber auch schon den sog. Mainstream erreicht. Dies wird durch die unzähligen Monographien dokumentiert (Han und Kamber 2001; Kimball und Caserta 2004; Runkler 2010; Chakrabarti 2009). In diesem Abschnitt werden wir uns nun die grundlegenden Vorgehensweisen während des ETL Prozesses ansehen und dabei besonderes Augenmerk der Verwendung der NCD schenken.

Datenintegration Die Integration von Daten aus verschiedenen Datenquellen bereitet immer wieder Schwierigkeiten. Besonders deutlich wird dies am Beispiel biomedizinischer Daten. Hier treten immer wieder große Qualitäts- sowie strukturelle und lexikalische Unterschiede auf. Konkret bedeutet dies, dass ähnliche Daten weder in derselben Struktur, noch in derselben Codierung vorliegen. Eine Adresse würde in so einem Fall in einem Datensatz über mehrere Felder (Name, Vorname, Anschrift, PLZ, etc.) verteilt und vollständig ausgeschrieben (Schreiberstraße) und in einem anderen Datensatz als eine zusammenhängende Zeichenkette mit Abkürzungen (Schreiberstr.) gespeichert. Besonders deutlich werden die Schwierigkeiten, wenn man in Betracht zieht, dass im zweiten Fall auch eine beliebige Reihenfolge dem einzelnen Datenelement (also die Anschrift vor dem Namen oder Name und Vorname vertauscht) erlaubt, was im ersten Fall nicht möglich ist. Zieht man nun noch Dinge wie UTF-8 vs. ASCII-Codierung mit in Betracht, so bekommt man einen Eindruck von der Komplexität der Aufgabe. Christen (2007) nennt nicht umsonst dies den Flaschenhals eines Record-Linkage Systems. Betrachtet man gängige Techniken zum Zeichenkettenvergleich so benötigen diese $O(|\sigma_1| \cdot |\sigma_2|)$ Vergleiche für zwei Zeichenketten $\sigma_{\{1|2\}}$ (Elmagarmid et al. 2007). Lernverfahren können hier Verbesserungen bringen, jedoch ist dort das Training wiederum sehr aufwändig.

Neben dem Vergleich einzelner Datensätze kann auch der Vergleich aller Datensätze selbst zum Problem werden. Für zwei Datensätze A und B mit $|A| = |B| = 100,000$ würde eine erschöpfende Suche zu 10^{10} Vergleichen (die Vergleiche der einzelnen Datensätze selbst nicht eingerechnet) führen. Eine Möglichkeit, dies zu umgehen, ist das sog. *blocking*. Dabei werden Datensätze zu homogenen Blöcken zusammengefasst, wobei ein Vergleich nur noch zwischen Repräsentanten der einzelnen Blöcke stattfindet. Für das Feld Postleitzahl könnte man z.B. Felder mit identischer Postleitzahl zusammenfassen. Wäre ein Kandidat nicht mit dem Repräsentanten identisch, so ganz sicher auch nicht mit einem anderen desselben Blocks. Eine weitere Variante ist das sog. Canopies based clustering. Dabei werden nicht identische Blöcke betrachtet, sondern sich überlappende Gruppen, die durch ein sehr schnell zu berechnendes Verfahren gewonnen werden. Kann man einen Kandidaten einer dieser

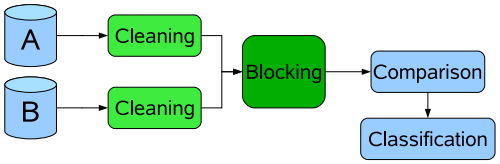


Abbildung 3.1: Der übliche Datenintegrationsprozess, wobei das Säubern der Daten vor dem Zuordnen der Daten zu den Feldern kommt.

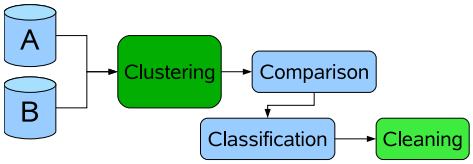


Abbildung 3.2: Der NCD-basierte Datenintergrationsprozess. Hier findet das Zuordnen der Daten zu den Feldern auf den verrauschten Daten und das Säubern dieser Daten mit dem Wissen der korrekten Zuordnung statt.

Gruppen zuordnen, so beginnt der Vergleich mit dem aufwändigeren und detaillierten Vergleichsverfahren (McCallum et al. 2000).

Vorgehensweise Der Vergleich zweier Zeichenketten mit Hilfe der NCD kann als Algorithmus mit linearer Zeitkomplexität implementiert werden. Dies bedeutet einen signifikanten Geschwindigkeitszuwachs verglichen mit den oben genannten Zahlen. Trotz dieses Leistungsgewinns kommt man, allein aufgrund des quadratischen Aufwands beim Vergleich jeder Zeichenkette mit jeder anderen, nicht um Blocking oder andere clusterbasierte Verfahren herum. Wir werden uns jedoch nur auf den einfachen Vergleich mit Hilfe der NCD konzentrieren.

Das Standardverfahren, beschrieben von Christen und Goiser (2007) und hier dargestellt in Abbildung 3.1, säubert (cleaning) die Daten vor dem eigentlichen Blocking und dem Vergleich. Dies ist notwendig, da sowohl der Zeichenkettenvergleich als auch das Abstandsmaß stark von der Domäne abhängen, in der sie verwendet werden. Man muss also im Voraus wissen, um was für eine Zeichenkette es sich handelt. Im Fall struktureller Homogenität kann es notwendig werden, Felder zu trennen oder zusammenzufügen. Betrachtet man dieses Vorgehen aus Sicht des gesamten Prozesses, so ergeben sich einige Nachteile: Als Erstes fehlt für eine erfolgreiche Bewertung zusammengehöriger Felder das Wissen um deren Inhalt. Des Weiteren hat eine Fehlentscheidung zu diesem Zeitpunkt weitreichende Folgen für den ganzen Prozess.

Wir werden uns nun ein alternatives Vorgehen, basierend auf der NCD, ansehen. Es wird in Abbildung 3.2 dargestellt. Bei diesem Prozess beginnen wir mit dem Vergleich der Zeichenketten. Das Clustering – der erste Schritt – wird nur aus Leistungsgründen eingeführt. Sind zwei Datensätze ρ_i und ρ_j gegeben, so werden beim Vergleich alle Felder der Datensätze aneinander angehängt, sodass man zwei große Zeichenketten σ_i und σ_j erhält. Wie wir in Abschnitt 3.2.1 sehen werden, hat die Reihenfolge, in der die Felder angeordnet sind, keinen Einfluss auf das Vergleichsergebnis. Aufbauend auf den zwei Zeichenketten können wir nun einen Vergleich mit Hilfe der NCD durchführen $d_{\mathrm{NCD}}(\sigma_i, \sigma_j) = d_{ij}$, der einen Ähnlichkeitswert liefert. Dieser Wert kann verwendet werden, um die Datensätze zu säubern. Auf diese Weise wird der Einfluss des Säuberns gering gehalten, jedoch das Gesamtergebnis verbessert.

Um zu entscheiden ob zwei Zeichenketten ähnlich genug sind, um eine einzige Entität zu repräsentieren, vergleicht man den gewonnenen Ähnlichkeitswert mit einem Grenzwert. Wird dieser unterschritten $d_{\mathrm{NCD}}(\sigma_i, \sigma_j) < t$ (es handelt sich ja um eine Distanz) geht man von identischen Einheiten aus. Eine besondere Herausforderung stellt die Wahl des Grenzwerts t dar. Er sollte klein genug gewählt werden, um ähnliche, jedoch nicht identische Entitäten, durchzulassen, aber gleichzeitig groß genug, um tatsächlich identische, jedoch mit kleinen Unterschieden behaftete Einheiten, auch als identisch zu erkennen. Im Information Retrieval kennt man dieses Problem als Gleichgewicht zwischen Precision und Recall (Baeza-Yates und Ribeiro-Neto 1999). Betrachtet man identische Einheiten als Ausreißer so hilft die Statistik in diesem Fall weiter: Die Ausreißer liegen signifikant näher an der Vergleichsentität als alle anderen. Die Umgebung der betrachteten Entität ist also dünn besiedelt oder, anders formuliert, der Abstand liegt außerhalb eines statistischen Vertrauensbereichs. Sind die statistischen Eigenschaften (Mittelwert μ und Varianz σ) der Verteilung der Abstände bekannt, ist es sehr leicht, einen guten Grenzwert

$$t = \mu - c\sqrt{\sigma}$$

zu bestimmen. In diesem Fall ist $c = \Phi^{-1}(\alpha)$ der Wert der Verteilungs-Quantil-Funktion über den Distanzwerten an der Stelle α. α bestimmt also den Vertrauensbereich. Abbildung 3.3 zeigt die Distanz einer Entität zu allen anderen. Blau ist der Mittelwert und Rot der Grenzwert. Wie man klar sehen kann, gibt es nur sehr wenige Entitäten mit Abständen in der Nähe des Grenzwerts.

Anwendungsszenario

Die zentrale Problemstellung, die wir im Folgenden diskutieren werden, ist die Integration großer, unstrukturierter und heterogener Datenquellen aus dem medizinischen Umfeld. Dabei sollte ein einfach zu implementierender und effizienter Algorithmus Anwendung finden. Er sollte auch mit Daten unterschiedlichster Formate umgehen können. Da die verwendeten Datenquellen – von Microsoft Excel bis hin zu Oracle Datenbanken können viele verschiedene Typen vorhanden sein – meist kein gemeinsames Identifikationsmerkmal für die einzelnen Entitäten haben, gilt es ein Verfahren zu finden, dass anhand der Ähnlichkeit Entitäten zuordnen kann.

Experimente In zwei Experimenten werden wir Einblicke in die Funktionsweise der vorgestellten Methode bekommen. Im ersten Test arbeiten wir mit zwei identischen Datensätzen, wobei Parameter wie die strukturelle oder die lexikalische Heterogenität variiert werden. Im zweiten Test betrachten wir zwei unterschiedliche Datensätze, die zugeordnet werden sollen.

Das erste Experiment besteht aus 241 Mitarbeiterdatensätzen der Informatik Fakultät der Universität Stuttgart. Er beinhaltet Daten wie Name, Raumnummer, Telefonnummer und Institutsschlüssel. Der Datensatz enthält eine klare Struktur und ist

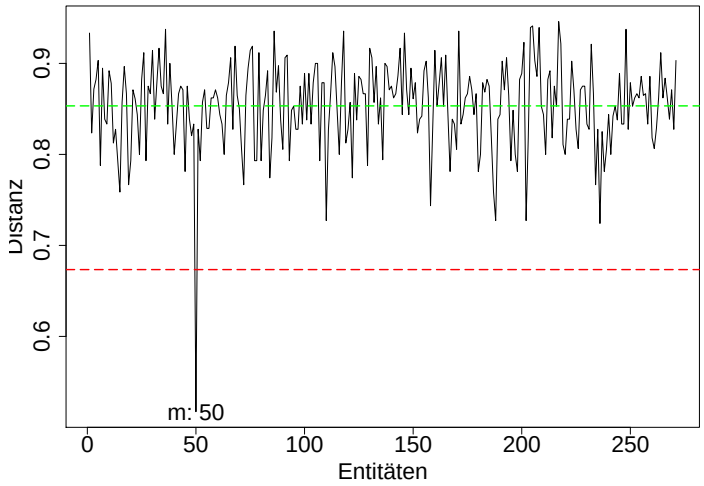


Abbildung 3.3: Die Distanzen einer Entität zu allen anderen. Zu einer gegebene Entität wurde die Distanz zu allen anderen gemessen. Der y-Wert gibt die gemessene Distanz zwischen der Entität und der, jeweils dem x-Wert entsprechenden wieder. Der grüne Strich entspricht dem Mittelwert aller Distanzen und der rote Strich dem untere Ende des Vertrauensbereichs. Die Werte, die jenseits der roten Linie liegen, gelten als Indiz dafür, dass es sich bei der korrespondierenden Entität um eine Dublette handelt.

frei von jeglichen syntaktischen Mehrdeutigkeiten. Aufgrund der einfachen Struktur ist es ein Leichtes, die Reihenfolge der Felder oder den Grad des Rauschens innerhalb der Felder zu variieren und zu beobachten, wie das Verfahren mit den geänderten Gegebenheiten umgeht.

Das zweite Experiment besteht aus zwei unterschiedlichen Tabellen, die jedoch teilweise identische Entitäten beinhalten. Ein Datensatz stammt aus dem Tumorzentrum eines Stuttgarter Krankenhauses (etwas mehr als $6,000$ Entitäten) und das andere aus einem Krebsregister in Stuttgart (ca. 600 Entitäten). Beide Datensätze enthalten die Adressen der behandelnden Ärzte. Sie unterscheiden sich sehr stark in Aufbau und Struktur und es gibt keinen gemeinsamen Schlüssel, der die jeweilige Entität eindeutig identifiziert. Darüber hinaus sind fast alle Freitextfelder, sodass Eingaben wie Telefonnummer oder Adresse sich teilweise gravierend (auch innerhalb derselben Tabelle) unterscheiden.

Mitarbeiterdaten In diesem Experiment betrachten wir, wie das Verfahren mit zunehmender lexikaler und struktureller Heterogenität umgeht. Dazu nehmen wir die Tabelle und erstellen eine Kopie, die mit Rauschen versehen ist. Wir suchen dann in der Kopie nach den Entitäten der Original-Tabelle.

Um die Tabelle mit dem Rauschgrad α zu versehen, wurde jedes einzelne Zeichen mit Wahrscheinlichkeit α durch ein zufälliges Zeichen ersetzt. Für $\alpha = 0.5$ wird im Schnitt jedes zweite Zeichen ersetzt.

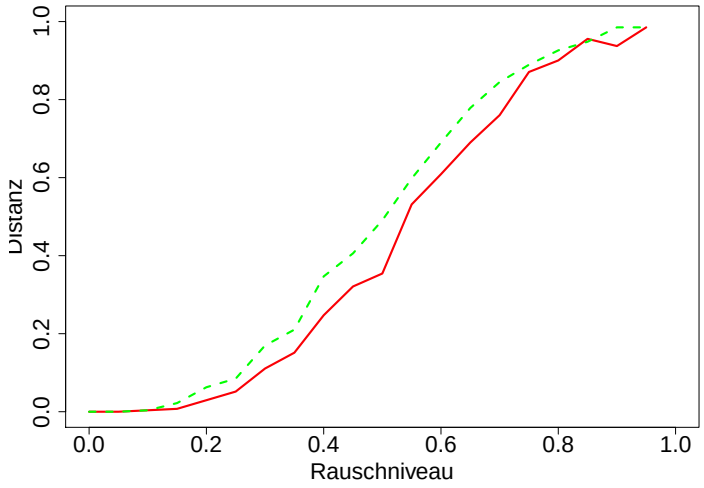


Abbildung 3.4: Die Entwicklung der Fehlerrate mit zunehmendem Rauschen. Die durchgezogene (rote) Linie repräsentiert dabei die Fehlerrate ohne strukturelle Änderungen und die doppelt gepunktete (grüne) Linie die Fehlerrate mit zwei Feldvertauschungen. Wie man sehen kann zeigt sich die NCD als ausgesprochen robust gegenüber strukturellen Änderungen, da sich die Entwicklung der Fehlerraten im Wesentlichen entspricht.

Zu jedem α berechnen wir eine verrauschte Kopie des Datensatzes. In dieser bestimmen wir dann die zueinander gehörenden Entitäten und können so ein Maß für die Robustheit des Verfahrens bezüglich des Rauschens bestimmen. Hierfür berechnen wir zu jeden Rauschgrad Precision P und Recall R

$$R = \frac{r_m}{r_r} \qquad \text{and} \qquad P = \frac{r_m}{m}$$

wobei m die Anzahl der zugeordneten Entitäten ist, r_m die der relevanten (also der „Treffer“) zugeordneten und r_r die aller relevanter (da es jede Entität nur einmal in diesem Datensatz gibt, ist $r_r = 1$) Entitäten ist. Abbildung 3.5 zeigt wie sich Precision und Recall mit zunehmendem Rauschen verändern.

Die Normalisierte Kompressionsdistanz hat für die hier vorgestellte Aufgabe einen großen Vorteil, sie ist weitestgehend robust gegenüber strukturellen Änderungen. Um einen Eindruck zu bekommen wie gut die NCD mit dieser Art von Änderung umgehen kann, haben wir Felder im Datensatz vertauscht. Der originale Datensatz bleibt dabei natürlich unangetastet. Nun beobachten wir, wie sich Precision und Recall ändern. Für ein gängiges Verfahren würde so eine Änderung eine manuelle Anpassung an dem Verfahren selbst nach sich ziehen. Die NCD jedoch scheint sehr gut damit umgehen zu können. Abbildung 3.6 zeigt den Einfluss von zwei Feldvertauschungen auf Precision und Recall. In Abbildung 3.4 wird der kombinierte Einfluss (Feldvertauschung und Erhöhung des Rauschens) auf P und R dargestellt. Die durchgezogene Linie repräsentiert dabei die Fehlerrate ohne strukturelle Änderungen und die gepunktete die

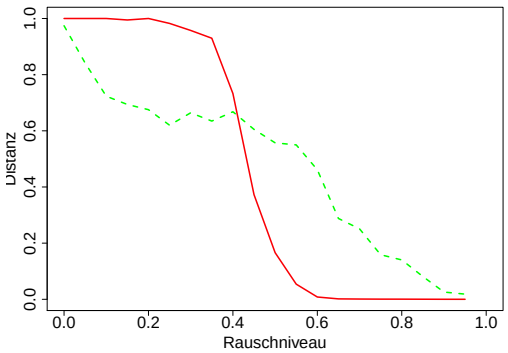


Abbildung 3.5: Die Entwicklung von Precision und Recall mit zunehmendem Rauschen. Die durchgezogene (rote) Linie repräsentiert dabei die Precision und die gepunktete (grüne) den Recall. Bei diesem Diagram wurden die Distanzen ohne die Vertauschung eines Feldes gemessen.

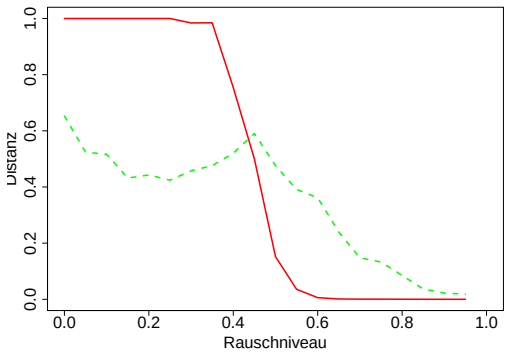


Abbildung 3.6: Die Entwicklung von Precision und Recall mit zunehmendem Rauschen und dem Vertauschen eines Feldes. Die durchgezogene (rote) Linie repräsentiert dabei die Precision und die gepunktete (grüne) den Recall. Der Feldvertausch schlägt sich hauptsächlich in der Precision nieder. Da, gerade bei niedrigen Fehlerraten, um einen vollständigen Recall zu erhalten, eine verminderte Precision toleriert wird. Bei größeren Fehlerraten ist auch ein verminderter Recall akzeptabel, was wiederum die Precision erhöht.

Fehlerrate mit einer, die doppelt gepunktete Linie mit zwei Feldvertauschungen. Die Abbildungen legen nahe, dass der Einfluss struktureller Änderungen gegenüber der Fehlerrate eher gering ist, besonders wenn man berücksichtigt, wie groß die Änderung an den Daten ist.

Ärztedaten Das zweite Experiment besteht aus Kontaktinformationen von Ärzten im Großraum Stuttgart. Es beinhaltet zwei Tabellen, eine aus der Datenbank eines Tumorzentrums und eine aus der Datenbank eines Krebsregisters. Die Inhalte der Tabellen überlappen sich teilweise, jedoch gibt es in jeder der beiden Tabellen Einträge (Ärzte) und Felder, die in der anderen nicht vorkommen. Der Einfachheit halber betrachten wir nur die ersten 100 Datensätze der Schnittmenge (bezogen auf die Entitäten, nicht die Felder) beider Tabellen. Als Referenztabelle betrachten wir die unveränderten Daten des Tumorzentrums und versuchen, diesen die Daten des Krebsregisters zuzuordnen. Diese Versuchsanordnung wurde gewählt, um eine einfache und übersichtliche visuelle Darstellung der Ergebnisse zu gewährleisten.

Die Struktur der beiden Tabellen ist stark heterogen. Die Tabelle des Krebsregisters beinhaltet folgende Felder:

`Code, CID, Salutation, Name, Field of Work, Street, Town, Status, Family Physician`

Das Tumorzentrum hat folgende Felder:

`TID, Date_edit, User_edit, ID_SAP, Zip, Country, Town, Title, Street, Name, Salutation, GivenName, Tel, Fax, ID_int, ...`

Wie man sehen kann enthalten die Tabellen unterschiedliche Felder für dieselben Inhalte. Einmal gibt es das Feld `Name` und dann die entsprechenden Felder `Name`, `Givenname` und `Titel`. Darüber hinaus müssen diese Felder nicht immer dieselben Daten enthalten. Besonders bei ausländischen Titeln kann es vorkommen, dass zum deutschen Titel `Dr. med.` noch weitere Informationen kommen oder diese zum Namen hinzugefügt werden (`Mr. XXXX YYYY (Dr. med.)`). Des Weiteren gibt es noch Abkürzungen oder andere Schreibweisen:

- `Dres. med. Maier/Müller/Schultze` vs. `D. Schultze & C. Müller`
- `CA PD Dr. med. Andrea Müller` vs. `Prof. Andrea Mueller`
- `Dr.-Medic/IM Temeschburg Erich-Maria Schultze` vs. `Herr Erich-Maria Schultze`

Um all diesen Schwierigkeiten Herr zu werden, bedarf es in einem klassisch strukturierten Prozess eines großen manuellen Aufwands. Mit Hilfe der hier beschriebenen Methode lässt sich dieser auf ein Minimum reduzieren.

In der hier beschriebenen Umgebung haben wir 100 Entitäten einer Testmenge Entitäten einer Referenzmenge zugeordnet. Wir haben die Daten in keiner Weise vorverarbeitet und haben uns ausschließlich auf den Vergleich der unveränderten Zeichenketten beschränkt. Die Ergebnisse zeigten, dass 80% der Entitäten korrekt identifiziert wurden. Für eine vollständig automatische Datenintegration ist dieser Wert zwar niedrig, verglichen mit dem Aufwand, den ein klassisches Vorgehen mit sich bringen würde, jedoch durchaus beachtlich.

Zusammenfassung

Im Rahmen dieses Anwendungsfalles haben wir uns mit der Zuordnung von Entitäten beschäftigt. Ziel war es, möglichst ohne menschliches Eingreifen korrespondierende Einheiten in unterschiedlichen Datensätzen zu erkennen. Hierzu verwendeten wir die NCD, die auf die unveränderten Datensätze angewendet wurde. Die Ergebnisse waren erstaunlich. Die NCD zeigte ein hohes Maß an Robustheit gegenüber Rauschen und struktureller Heterogenität.

Gerade der Einsatz der NCD im Rahmen von Aufgaben, bei denen wenig Wissen über die Daten vorhanden ist, scheint große Vorteile zu bringen. Wir konnten trotz

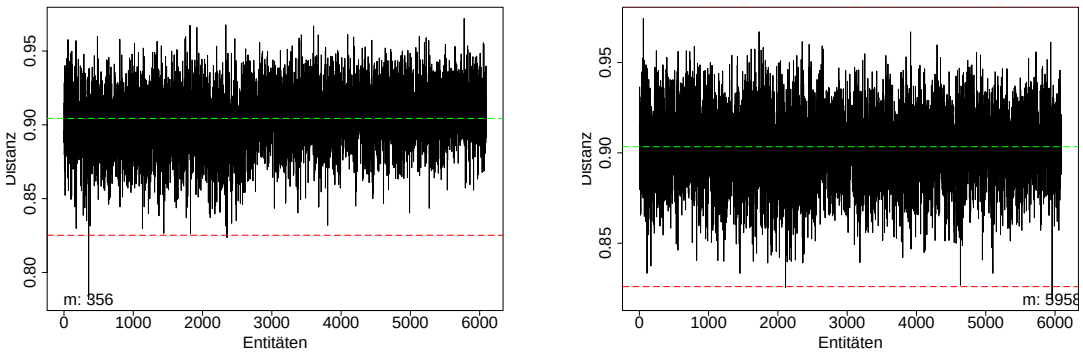


Abbildung 3.7: Die Distanzen des Referenzdatensatzes zu zwei Elementen (Element 10 und Element 50) aus der Testmenge. Das linke Bild zeigt alle Distanzen zum ersten Datensatz, das zweit alle Distanzen zum Zweiten. In beiden Bildern ist der Mittelwert (gestrichelte grüne Linie) und der Vertrauensbereich (gestrichelte rote Linie) eingezeichnet. Die Distanzen, die diesen Bereich verlassen, gelten als Treffer. Die gefundenen Treffer (hier 356 und 5958) entsprechen den gesuchten Entitäten, wobei es sich dabei um Entitäten aus dem Referenzdatensatz handelt, weshalb die Nummerierung sich nicht entspricht.

stark unterschiedlicher Datensätze und ohne Vorwissen oder Vorverarbeitung Erkennungsraten von über 80% erreichen.

Einen Aspekt, der in diesem Anwendungsfall keine Verwendung fand, jedoch für die praktische Arbeit von großer Bedeutung ist, ist die Verfügbarkeit hochperformanter Algorithmen. Die NCD lässt eine Verarbeitung in Linearzeit zu, was im Gegensatz zu den üblicherweise quadratischen Algorithmen deutliche Geschwindigkeitssteigerungen erwarten lässt.

Nun da wir uns einen Anwendungsfall der NCD angesehen haben, werden wir uns dieses Verfahren genauer betrachten. Der folgende Abschnitt wird durch Experimente Einblicke in die Funktionsweise der NCD geben.

3.2.2 Experimente

In zahlreichen praktischen Arbeiten konnte die NCD bereits ihre Leistungsfähigkeit demonstrieren. Das Fundament dieses Verfahrens scheint durch die Kolmogorovkomplexität ausreichend motiviert. Was noch fehlt, ist eine Verbindung zwischen der praktischen Anwendung und der Theorie, etwas das es uns ermöglicht zu verstehen, warum die NCD so erfolgreich ist.

In diesem Abschnitt werden wir uns den Fragen nähern, wie die NCD Muster erkennt, was genau die Merkmale sind, anhand deren Ähnlichkeiten bestimmt werden und wo genau die Stärken und Schwächen des Verfahrens liegen. Das alles führt uns dann auch zu der Frage, wie man die NCD verbessern und verallgemeinern kann.

Die bekanntesten Beispielanwendungen der NCD sind:

1. Sprachengruppierung,
2. Autorengruppierung und
3. Säugetiergruppierung.

All diese Anwendungen basieren auf diskreten Daten,[3] diese beziehen in diesen konkreten Fällen ihre Semantik aus der Abfolge von Symbolgruppen, vereinfacht könnte man diese Gruppen auch Worte nennen. Für die Interprätierbarkeit ist dies eine kritische Eigenschaft. Wir werden uns deshalb im Rahmen dieser Experimente auf die beiden ersten Anwendungsfälle konzentrieren. Dabei ist ein klarer semantischer Zusammenhang zwischen den einzelnen Zeichenketten erkennbar. Dies gilt zwar auch für DNA-Sequenzen, der Zusammenhang ist dort jedoch nicht ganz so offensichtlich. Die hier präsentierten Ergebnisse basieren zum großen Teil auf den Arbeiten von Burkovski et al.

Sprachengruppierung

Zahlreiche Sprachen ähneln sich. Wie stark sie sich ähneln hängt davon ab, ob sie aus derselben Sprachfamilie stammen oder nicht. Sprachen einer Familie haben oft einen ähnlichen Wortschatz. Viele Wörter haben einen gemeinsamen Wortstamm. Das Wissen um diese Eigenschaften kann genutzt werden, um Sprachen zu gruppieren. Wenn man einen Text hat, der jeweils in zwei verschiedenen Sprachen geschrieben wurde, so kann man davon ausgehen, dass man mehr gemeinsame Zeichenketten findet, falls die Texte aus der gleichen Sprachfamilie stammen. Dies bedeutet aber auch, dass eine gemeinsame Kompression effizienter bei ähnlichen Sprachen wäre. Benedetto et al. (2002) versuchen dies in ihrer Arbeit auszunutzen, indem sie Texte anhand ihres Kompressionsgrads in Gruppen ordnen.

Als Basis für die Gruppierung von Sprachen dient die Menschenrechtscharta der Vereinten Nationen ("Declaration of Human Rights„ – `http://www.un.org`). Sie wurde in 52 Sprachen übersetzt. Ziel ist es nun, eine Gruppierung zu finden, die unserem intuitiven Verständnis für die Ähnlichkeit der einzelnen Sprachen untereinander entspricht. Als Referenz gilt dabei Ethnologue[4].

NCD Experiment Unser erster Schritt im Rahmen dieser Experimente wird es sein, die gewonnenen Ergebnisse nachzuvollziehen. Wir werden zu diesem Zweck den Versuchsaufbau von Benedetto et al. (2002) übernehmen, wobei wir das Distanzmaß

$$d_S(x,y) = \frac{(\Delta_{xy} - \Delta_{yy})}{\Delta_{yy}} + \frac{(\Delta_{yx} - \Delta_{xx})}{\Delta_{xx}}$$

[3] Wie bereits in der Einleitung beschrieben, betrachten wir Daten mit einem abzählbaren Wertebereich als diskret.

[4] Eine Enzyklopädie aller 6.909 noch lebenden Sprachen der Erde `http://www.ethnologue.com/`

hierbei ist $\Delta_{xy} = C(x+y) - C(x)$ die Differenz der Kompressionsgröße von x und der Konkatenation $x + y$, durch die NCD ersetzen werden. Die Distanzmatrix $(d_{i,j})$, wobei $d_{ij} = d_S(x_i, x_j)$ ist, nutzen wir als Eingabe für ein hierarchisches und ein phylogenetisches Clustering (Fitch und Margoliash 1967).

Als Korpus verwenden wir alle Sprachversionen der Menschenrechtscharta. Aus diesen entfernen wir sämtliche HTML Tags, Zeilenumbrüche und alle unnötigen Leerzeichen. Außerdem werden alle Dokumente von UNICODE zu ASCII transformiert. Dies gewährleistet, dass die NCD ausschließlich auf den rohen Textdaten arbeitet. Für die Berechnung der Distanzmatrix, also der Matrix deren Einträge die paarweisen Distanzen zwischen den einzelnen Dokumenten bilden, verwenden wir die Normalisierte Kompressionsdistanz (Definition 2.13).

$$d_{\mathrm{NCD}}(x, y) = \frac{C(xy) - \min\{C(x), C(y)\}}{\max\{C(x), C(y)\}}.$$

Hierbei verwenden wir im ersten Term im Zähler jeweils die kompletten Dokumente zur Konkatenation. Passend hierzu stellen wir den Such- und Vorschaupuffer so ein, dass die Dokumente vollständig hineinpassen. Da *gzip* [5] standardmäßig eine Puffergröße von $32kB$ verwendet, die Texte im Schnitt jedoch $10kB$ bis $20kB$ groß sind, wird der Puffer auf $40kB$ erweitert. Zur Berechnung der Größe der komprimierten Datei $C(x)$ bestimmen wir die Anzahl der verwendeten LZ77-Tupel[6]. Dies entspricht der tatsächlichen Größe, jedoch skaliert um einen konstanten Faktor. Hierfür verwenden wir eine von Burkovski et al. entwickelte Implementierung des LZ77-Algorithmus. Dieser erlaubt es, Parameter wie zum Beispiel die Puffergröße frei zu wählen. Darüber hinaus gibt er Zugriff auf Interna des Verfahrens (wie zum Beispiel die Tupel selbst oder die darin referenzierten Zeichenketten). Den eigentlichen Baum (die Darstellung des Clusterings) bestimmen wir mit Hilfe der Statistikprogrammiersprache R (R Development Core Team 2010) und dem darin enthaltenen Paket für das hierarchische Clustering. Zur Gruppierung verwenden wir die *complete* Clustering Methode. Das Ergebnis finden wir in Abbildung 3.8. Wer sich mit der ethnologischen Gruppierung der Sprachen auskennt, ist bei der Interpretation der Abbildung klar im Vorteil. Dem wissenden Leser wird dabei auffallen, dass weder Isländisch noch Färöisch richtig eingeordnet sind. Sie sollten im germanischen Zweig liegen. Versuchen wir dieselben Daten mit Hilfe des phylogenetischen Clustering von Fitch und Margoliash zu gruppieren, so erhalten wir Abbildung 3.9. Diese Gruppierung stimmt exakt mit der von Benedetto et al. (2002) überein.

Dieser erste Versuch legt nahe, dass die Clusteringmethode einen erheblichen Einfluss auf das Versuchsergebnis hat. Die uns soweit zur Verfügung stehenden Informationen reichen jedoch noch nicht aus, Näheres zur kompressionsbasierten Musterer-

[5] *gzip* ist eine populäre Implementierung des LZ77 Algorithmus von Ziv und Lempel (Wikipedia 2010c, für weitere Informationen).

[6] Wir werden im Folgenden immer wieder Tupel und den von ihnen referenzierten Abschnitt gleichsetzen. Wenn wir also von der Tupellänge sprechen meinen wird damit nicht die Summer der Bytes von Zeichen und Referenz sonder die Länge des Abschnitts auf den die Referenz zeigt.

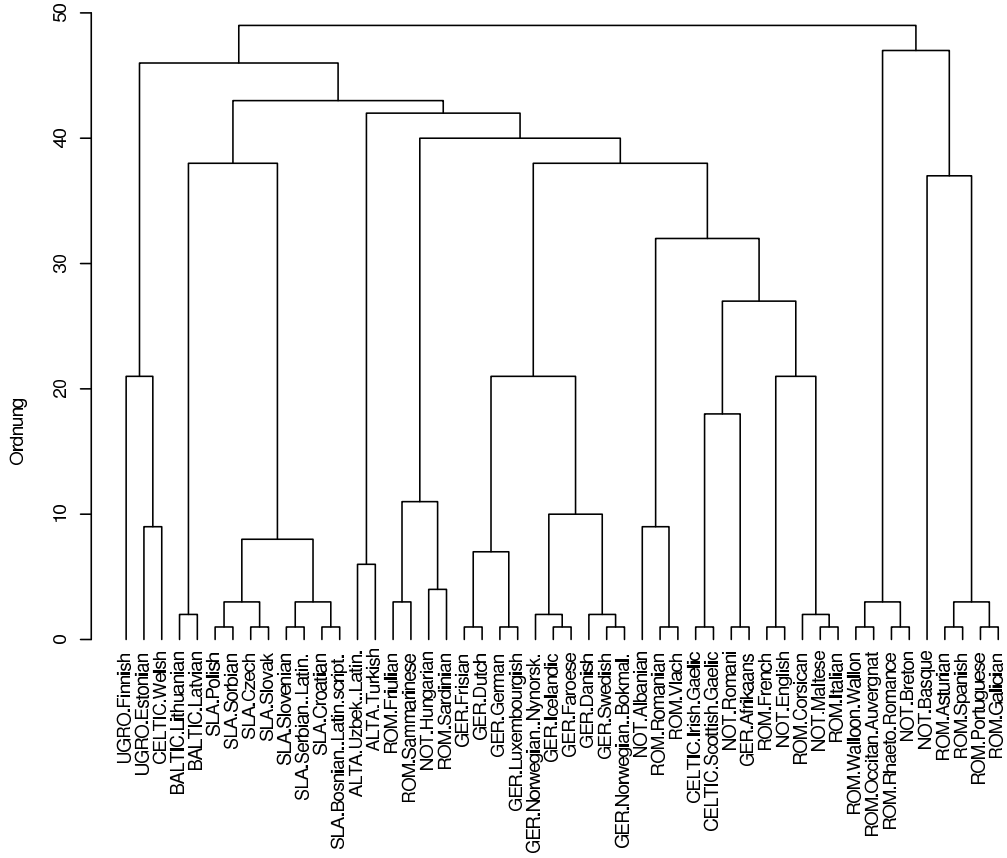


Abbildung 3.8: Der Sprachenbaum erstellt mit NCD_{gzip} und der hierarchischen Clusteringmethode *complete* aus R. Die Sprachen werden, bis auf Isländisch und Färöisch, korrekt den jeweiligen Sprachgruppen zugeordnet. Bessere Ergebnisse erhalten wir durch die Verwendung des phylogenetischen Clusterings.

kennung mit Hilfe der NCD in Erfahrung zu bringen. Wir haben zwar gesehen, dass die NCD gut arbeitet aber nicht warum sie dies tut. Diese Lücke werden wir nun schließen, indem wir uns die Verteilung der einzelnen LZ77-Tupel ansehen. Wir werden uns dabei auf einen Vergleich der Dateien folgender Sprachen beschränken: *Deutsch*, *Englisch*, *Luxemburgisch*, *Spanisch* und *Slowenisch*. Diese Sprachen stehen jeweils für eine der drei Sprachgruppen *germanisch* (Deutsch, Englisch und Luxemburgisch), *slawisch* (Slowenisch) und *romanisch* (Spanisch). Englisch nimmt dabei eine Sonderrolle ein, da es auch sehr große Ähnlichkeit mit den romanischen Sprachen hat. Die nachfolgenden Visualisierungen und Statistiken zur Tupelverteilung entsprechen denen, die mit anderen Sprachen derselben Sprachgruppen zu erwarten sind.

Kompressionsvisualisierung Für ein besseres Verständnis der Kompression mit Hilfe des LZ77-Algorithmus und dessen Einfluss auf die Mustererkennung werden wir uns ansehen, welche Abschnitte komprimiert werden und auf welches Dokument die

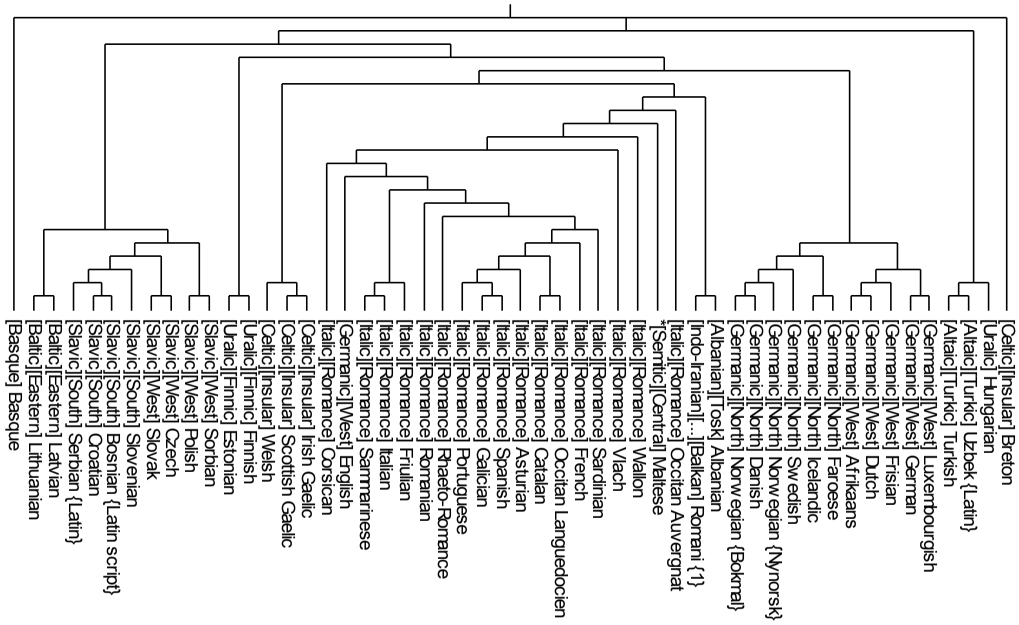


Abbildung 3.9: Der Sprachenbaum erstellt mit NCD_{gzip} und der Fitch-Margoliash Clustering-methode des PHYLIP Pakets. Die Sprachen werden korrekt (entsprechend der Arbeit von Benedetto et al.) den jeweiligen Sprachgruppen zugeordnet.

-	Englisch	Luxemburgisch	Spanisch	Slowenisch
Deutsch	0.9126506024	0.8606481481	0.9146586345	0.9257028112

Tabelle 3.3: Distanzen des deutschen Dokuments zu den anderen Dokumenten.

Referenzen zeigen. Dazu verwenden wir eine neuartige Visualisierungsmethode. Jedem Zeichen wird ein Pixel zugeordnet. Da jedes Zeichen Teil eines Tupels ist, bestimmen wir die Intensität des Pixels abhängig von der Tupellänge (lange Tupel werden kräftig gezeichnet, kurze eher blass). An der allgemeinen Farbintensität des Bildes kann man daher erkennen, ob eher viele kurze oder wenige lange Tupel gefunden wurden. Kräftig grüne Bilder deuten eine viele lange Tupelverteilung mit vielen langen Tupeln an wohingegen eher blasse Bilder auf eine Verteilung mit vielen kurzen Tupeln deutet.

Da wir uns für die Kompression der konkatenierten Dateien interessieren, werden wir auch noch die Tupel kennzeichnen, auf die aus dem konkatenierten Dokument verwiesen wird. Diese sind besonders interessant, da sie dazu führen, dass die gemeinsam komprimierten Dokumente eine geringere Größe haben als die Summe der Länge der einzelnen komprimierten Dokumente. Diesen Pixeln werden wir die Farbe Blau zuweisen. Zu jeder Fremdreferenz – also einer Referenz die aus dem konkatenierten Dokument auf ein Zeichen in dem anderen Dokument zeigt – gibt es auch noch das Zeichen, das referenziert wird. Dieses werden wir rot zeichnen. Ein gegebenes Zeichen

kann also drei Farben haben, Grün falls es ein Zeichen im Suchpuffer referenziert, das noch in der selben Datei liegt, Blau falls das referenzierte Zeichen in der (links-[7]) konkatenierten Datei liegt und Rot falls dieses Zeichen Teil einer referenzierten Zeichenkette, ist auf die aus der (rechts-) konkatenierten Datei ein Tupel zeigt.

Betrachten wir nun die Visualisierungen der Kompression für die bereits erwähnten vier Sprachen bezüglich der deutschen Sprache. Abbildung 3.10 zeigt, welche Tupel in den Texten referenziert werden. Man sieht deutlich, dass je ähnlicher zwei Dokumente sind, desto mehr Tupel werden aus der (links-) konkatenierten Datei referenziert. Ebenso ist zu erkennen, dass die durchschnittliche Tupellänge mit der Ähnlichkeit zunimmt.

Eine wichtige Frage in diesem Kontext ist die nach identischen Referenzen. Was geschieht, falls es für ein Tupel mehrere identische Referenzen gibt? Das Wort „und“ wäre zum Beispiel so ein Kandidat, der häufig wiederzufinden ist und jede Referenz die auf ein vorangegangenes „und“ verweißt wäre so gut wie die andere, man hätte also die freie Auswahl. Im Fall der Visualisierung werden wir uns darauf beschränken, die erste Möglichkeit als Basisabschnitt zu wählen und alle entsprechenden Referenzen auf diesen Abschnitt zeigen lassen. Welchen Einfluss dies auf die Kompression hat, werden wir im folgenden Abschnitt sehen.

Tupelverteilung Die Tupelverteilung ist von zentraler Bedeutung bei der Kompression. Je mehr Referenzen auf lange Abschnitte gefunden werden, umso größer ist der Kompressionsfaktor. Bei konkatenierten Dokumenten bedeuten Referenzen auf lange Abschnitte aus dem anderen Dokument eine große Ähnlichkeit. Dies wird besonders deutlich wenn man sich vor Augen hält, dass die Referenz mit der größt möglichen Länge auf das (links-) konkatenierte Dokument selbst zeigt und somit zu einer minimalen Distanz und einer größt möglichen Ähnlichkeit führt. Die naheliegende Frage in diesem Abschnitt ist nun ob man aus der Tupelverteilung allein schon Rückschlüsse auf die Ähnlichkeit von Dokumenten ziehen kann. Vergleich man mehrere Dokumente so stellt sich die Frage ob bei zwei semantisch ähnlichen Dokumenten mehr lange (und damit auch hoffentlich semantisch relevante) Zeichenketten referenziert oder eher nur kurze, möglicherweise weniger relevante Zeichenketten gefunden werden. Des weitere ist es interessant zu Wissen inwiefern sich die Tupelverteilung zweier ähnlichen Dokumenten von der zweier unähnlichen unterscheidet.

Bei der Betrachtung der Häufigkeit von Tupellängen, also den Längen der Abschnitte auf die die Referenz des Tupels verweisst, muss man darauf achten nicht einfach die Anzahl der Tupel miteinander zu vergleichen. Lange Tupel werden zwangsläufig seltener auftreten als kurze. Es ist daher nicht unbedingt erstaunlich in einem Text von 100 Zeichen zehn Referenzen auf Abschnitte der Länge fünf zu finden, wo-

[7] Wir werden hier zwischen dem *links*- und dem *rechts*-konkatenierten Dokument unterscheiden. Die Konkatenation fügt an ein gegebenes Dokument (repräsentiert als eine durchgehende Zeichenkette) rechts ein neues Dokument an. Das linkskonkatenierte Dokument ist somit das Dokument, an das ein neues angehängt wird. Daher ist das rechtskonkatenierte Dokument das, das angehängt wird.

hingegen zehn Referenzen auf Abschnitte der Länge zehn jedoch eine große Besonderheit wären. Um dies zu berücksichtigen, gewichten wir die Tupel mit der Länge des Abschnitts auf den sie referenzieren. Dies ist gleichbedeutend mit dem Zählen der Zeichen, die in den von den Tupeln referenzierten Abschnitten vorkommen.

Für das Verständnis der Normalisierten Kompressionsdistanz ist es von zentraler Bedeutung, wie viele Referenzen aus dem rechts-konkatenierten Dokument auf das links-konkatinierte zeigen. Erst dies führt dazu, dass die Konkatenation besser komprimiert werden kann als die einzelnen Dokumente alleine. In diesem Zusammenhang wurde bereits erwähnt, dass es durchaus einen Unterschied macht, welche Referenz (falls mehrere identische Möglichkeiten vorhanden sind) ausgewählt wird. Eine Referenz im konkatenierten Dokument wäre (nach bisheriger Argumentation) zu bevorzugen, da es die gemeinsame Dateigröße reduziert. Eine Referenz auf eine Zeichenkette im selben Dokument hat jedoch genau das selbe Ergebnis: Die gemeinsame Dateigröße reduziert sich um die Länge der Referenz. Hier nun eine kurze Beschreibung der zwei Extreme.

first-match Falls es zu einer Zeichenkette mehrere identische Referenzen gibt, so wählen wir die Erste aus. Diese liegt bei ähnlichen Dokumenten meistens im links-konkatenierten Dokument. Diese Variante bevorzugt also sogenannte Fremdreferenzen[8].

last-match In diesem Fall wählen wir die letzte der möglichen Zeichenketten welche auch bei ähnlichen Dokumenten oft im selben Dokument liegt. Liegt die gerade betrachtete Zeichenkette im rechts-konkatenierten Dokument so bedeutet das, dass die Referenz mit großer Wahrscheinlichkeit ebenfalls in diesem Dokument liegt. Diese Variante bevorzugt Eigenreferenzen[9].

Nachdem wir nun die zwei Vorgehensweisen bei der Wahl der Referenzen betrachtet haben, werden wir uns ansehen, wie stark sich der Unterschied auf die Tupelverteilung auswirkt. Abbildung 3.11(a) gibt die Tupelverteilung der unterschiedlichen Sprachen für die first-match Variante an, Abbildung 3.11(b) die für die last-match Variante.

Was als erstes an den Abbildungen auffällt, ist die starke Rechtsverschiebung, also die höheren Werte für größere Tupellängen, bei den Sprachen Luxemburgisch und Englisch. Dies zeigt an, dass diese Sprache deutlich mehr lange (mit Längen zwischen vier und acht für den englischen Text und zwischen vier und siebzehn für den luxemburgischen Text) Referenzen haben als die anderen Texte. Zusammen mit der größeren berechneten Ähnlichkeit deutet dies darauf hin, dass lange Tupel stark die Ähnlichkeit beeinflussen. Die last-match Variante verstärkt für den luxemburgischen Text dieses Bild noch.

Betrachtet man die gefundenen Referenzen, so wird der Zusammenhang zwischen Länge der Tupel und Ähnlichkeit der Texte nochmals verdeutlicht. In dem luxemburgischen Text findet man häufig deutsche Wörter wie zum Beispiel „Artikel 1“ oder

[8] Als *Fremdreferenzen* werden wir Referenzen bezeichnen, die nicht im selben Dokument liegen.
[9] Als *Eigendreferenzen* werden wir Referenzen bezeichnen, die im selben Dokument liegen.

-	|Wörter|	∅ Wortlänge
Deutsch	1686	6,25
Luxemburgisch	1951	5,49
Englisch	1788	5,01
Spanish	1969	5,21
Slowenisch	1546	5,71
Deutsch + Luxemburgisch	3637	5,84
Deutsch + Englisch	3474	5,66
Deutsch + Spanisch	3655	5,69
Deutsch + Slowenisch	3232	5,99

Tabelle 3.4: Die Statistik der unkomprimierten Dokumente.

Teilwörter, die mit deutschen Worten identisch sind. Im Englischen wird dies schon seltener. In den spanischen oder slowenischen Texten bilden sich die meisten Referenzen nur aus Zeichenkombinationen, wie zum Beispiel „a b“ oder „x. y“ (diese Fragmente ergeben sich an Wortenden beziehungsweise -anfängen) die zwar häufig vorkommen, für eine Ähnlichkeitsbetrachtung der Texte jedoch keine Bedeutung haben.

Etwas weniger anschaulich, dafür aber deutlich konkreter ist Tabelle 3.5. Dort kann man sich exakte Zahlenwerte zu verschiedenen Kenngrößen der Verteilungen ansehen. Als Vergleich dazu dient Tabelle 3.4, die sich auf die unkomprimierten Dokumente bezieht. Die dort angegebene Wortlänge bezieht sich auf die Länge der Wörter im Text. Interessant in diesem Zusammenhang ist, dass sowohl die durchschnittliche Tupellänge, als auch die durchschnittliche Wortlänge zwischen vier und sechs schwanken. Dies legt die Vermutung nahe, dass die meisten Tupel sich auf Wörter beziehen. Ein Blick auf die referenzierten Zeichenketten bestätigt diese Vermutung. Betrachtet man nur die Kompression einzelner Dokumente (also nicht die der Konkatenation wie in der NCD verwendet) sieht man klar, dass die meisten Tupel Worte oder Wortfetzen referenzieren.

Was die Tabellen und die Abbildungen nahelegen, scheint für Sprachen auch im Alltag zu gelten. Beim Lesen erkennt man eine fremde Sprache am leichtesten an den fremden Wörtern. Nicht klar jedoch ist, ob dies auch für Autoren, wie in unserem nächsten Experiment, gilt. Hier stellt sich die Frage, ob eine erfolgreiche Gruppierung auch nur die verwendeten Worte betrachtet oder ob in diesem Fall größere Textblöcke notwendig sind.

Autorengruppierung

Einen Text einem Autor zuzuordnen ist für einen Laien keine einfache Aufgabe. Auch Fachleute verwenden in solchen Situationen oft sehr komplexe Konzepte wie Genre, Schreibstil oder Stimmungsbilder, die einen Autor auszeichnen. Für die maschinelle Zuordnung von Texten zu ihren Autoren wären diese Konzepte zu wenig greifbar und

-	|Tupel|	∅ Tupellän.	|FR|	∅ FR-län.	∅ FR-gew.
Deutsch	2026	5,03	0	-	-
Luxemburgisch	2194	4,77	0	-	-
Englisch	1753	5,22	0	-	-
Spanisch	1953	5,27	0	-	-
Slowenisch	1996	4,28	0	-	-
Deutsch + Luxemburgisch	4078	5,10	255	3,90	135,7
Deutsch + Englisch	3690	5,27	182	3,51	106,0
Deutsch + Spanisch	3892	5,28	152	3,31	97,8
Deutsch + Slowenisch	3941	4,73	122	2,90	84,0

Tabelle 3.5: Die Statistik der Kompression von Dokumenten. Die Dokumente wurden mit der *last match* Variante komprimiert. Das „+“bedeutet, dass zwei Dokumente mit einander konkateniert werden. FR bezeichnet die Fremdreferenzen.

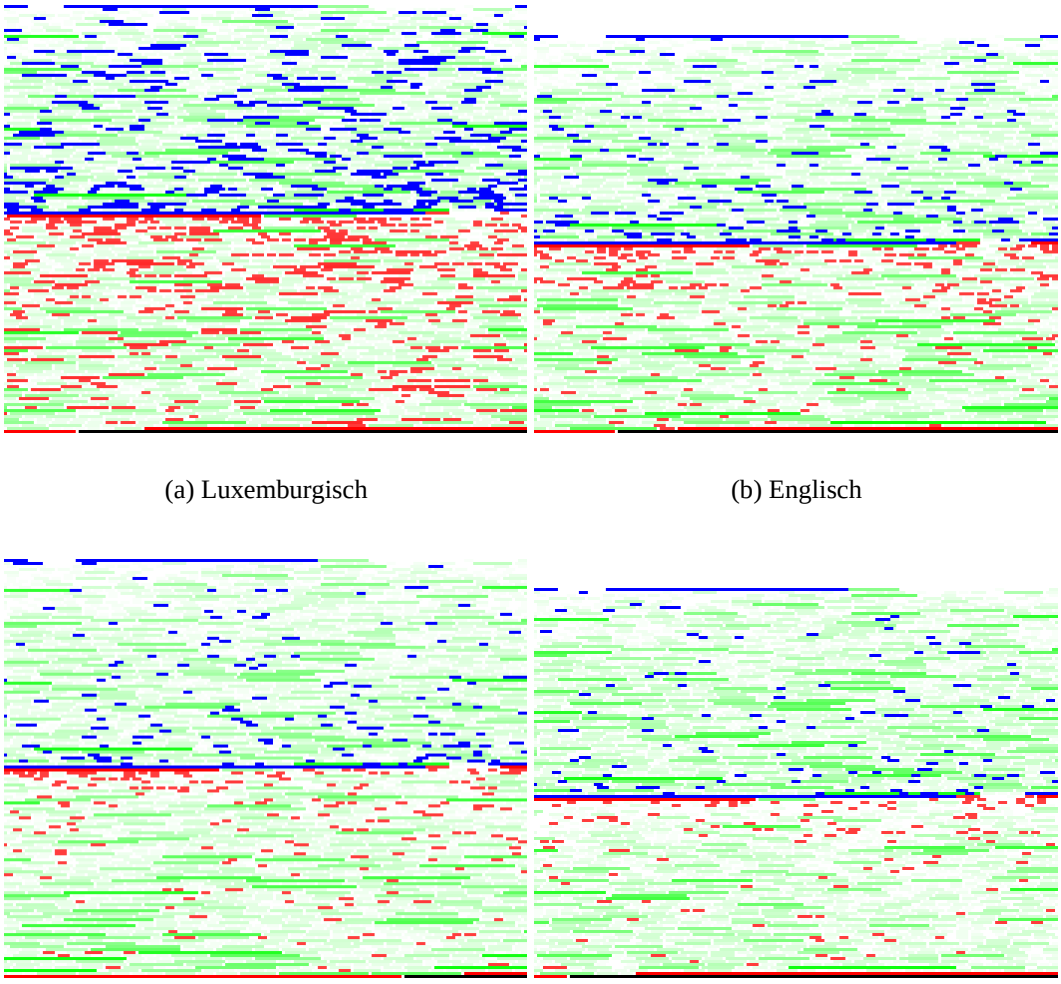
praktisch nicht zu formalisieren. Um so erstaunlicher, dass es Cilibrasi und Vitànyi (2005) gelungen ist, Autoren mit Hilfe der NCD anhand ihrer Texte zu gruppieren.

Im Rahmen dieses Experiments werden wir uns ansehen, welche Muster die NCD in diesem Fall erkennt. Was sind die Merkmale, die zu einer erfolgreichen Gruppierung von Autoren herangezogen werden und welchen Einfluss haben sie auf die Mustererkennung?

NCD Experiment Das Experiment sieht wie folgt aus: Wir haben einen Korpus bestehend aus den Büchern von I.S. Turgenew (Am Vorabend; Rudin; Väter und Söhne; Home of the Gentry), F.M. Dostoyevsky (Arme Leute; Der Idiot; Der Spieler; Schuld und Sühne), L.N. Tolstoi (Kindheit; Anna Karenina; Die Kosaken; Krieg und Frieden), N.W. Gogol (Geschichte des Streitfalls; Iwan Iwanowitsch gegen Iwan Nikiforowitsch; Taras Bulba; Die toten Seelen; Das Portrait) und M.A. Bulgakow (Der Meister und Margarita; Hundeherz; Die verhängnisvollen Eier). Die Werke sind frei zugänglich in Bibliotheken im Internet verfügbar (`http://www.lib.ru/`, `http://www.gutenberg.org/`). Cilibrasi und Vitànyi verwenden für ihre Berechnungen den *bzip2* Kompressor und erzielen damit eine fast perfekte Aufteilung nach Autoren. Wir werden jedoch hier einen anderen Fokus setzen und werden auch diesmal mit dem von Burkovski et al. implementierten LZ77-Algorithmus arbeiten um einen genauen Einblick in die Arbeitsweise der NCD zu erhalten.

Unser erstes Anliegen wird es sein, die Ergebnisse von Cilibrasi und Vitànyi zu rekonstruieren. Dazu wenden wir einerseits das *gzip* Verfahren und andererseits den LZ77 Algorithmus, von Burkovski implementiert, an. Die Algorithmen von *gzip* und LZ77 sind identisch, bei dem Versuch variiert ausschließlich die Puffergröße. Da die Dokumente teilweise sehr groß sind (bis zu $6MB$ im Fall von Krieg und Frieden) und der Suchpuffer von *gzip* auf $32kB$ begrenzt ist, ist der Anteil, in dem tatsächlich ein Vergleich der Dokumente stattfindet, sehr gering. Für die zwei kleinsten Dokumente

(jeweils ca. $160kB$) würden nur 10% der Gesamtgröße in den Vergleich einfließen. Um daher den Einfluss der Puffergröße so klein wie möglich zu halten, führen wir die Berechnungen auf zurechtgestutzten Dateien durch, d.h. wir betrachten nur die ersten n Bytes einer Datei. Tabelle 3.6 gibt Aufschluss über die Versuchsergebnisse in Abhängigkeit von n (dargestellt in Abbildung 3.12 wobei ein Fehler („Schuld und Sühne“) vorhanden ist). Dort ist klar erkennbar, dass *gzip* nie eine korrekte Gruppierung vornimmt, LZ77 und *bzip2* hingegen schon auf recht kleinen Dateien (LZ77 ab $85kB$ und *bzip2* ab $106kB$) perfekte Ergebnisse erzielen. Da es sich bei LZ77 und *gzip* um denselben Algorithmus handelt – der Unterschied liegt ausschließlich in der Puffergröße – legen die Ergebnisse nahe, dass Parameter wie die Puffergröße einen sehr großen Einfluss auf die Leistung des Verfahrens haben.



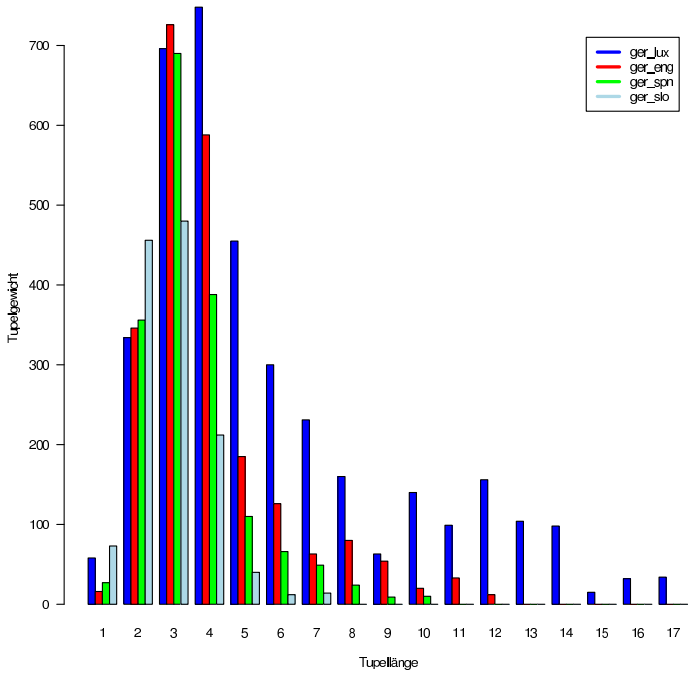
(a) Luxemburgisch

(b) Englisch

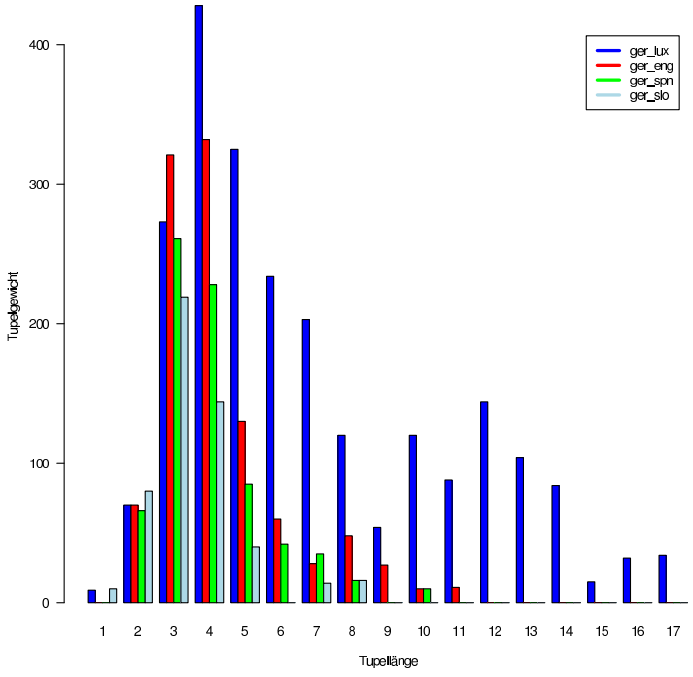
(c) Spanisch

(d) Slowenisch

Abbildung 3.10: Der jeweilige Text ist eine Konkatenation des deutschen Textes und des jeweils angegebenen. Die *grüne* Farbe zeigt Zeichen, die anhand des eigenen Textes komprimiert wurden (entweder in Form eines Tupels oder als Teil einer Referenz auf einen vorangegangenen Abschnitt). Die *rote* Farbe zeigt, welche Teile des Fremddokuments auf Abschnitte des Originaldokuments verweisen. Die *blaue* Farbe ist die Fremdreferenz im Originaldokument, die im Fremddokument komprimiert wurde. Die Intensität der Farbe deutet auf die Tupellänge hin. Zur besseren Darstellung betrifft die Intensitätskodierung nur die *grüne* Farbe.



(a) First-match Variante



(b) Last-match Variante

Abbildung 3.11: Die Bilder zeigen eine gewichtete Verteilung der Tupel. Deutlich zu sehen (*blau*) ist, dass ein großer Teil des *luxemburgischen* Textes im *deutschen* gefunden werden kann. Der *slowenische Text* hingegen hat weniger und durchschnittlich kürzere Tupel als bei anderen Dokumenten. Grundsätzlich legen die Bilder nahe, dass semantisch ähnlichere Dokumente auch eine Tupelverteilung haben, die stärker in den hohen Tupellängen präsent ist. Der Vergleich von First-Match und Last-Match zeigt, dass semantisch ähnlichere Dokumente mehr Fremdreferenzen besitzen als weniger ähnliche (der Unterschied zwischen den Kurven ist bei *luxemburgischen* weniger gravierend als bei *slowenisch*).

-	gzip		LZ77		*bzip2*		LZ77 Clustering	gzip Clustering	*bzip2* Clustering
16384B	16%	3	53%	10	37%	7		[Gogol, Turg.]	[Bulg.]
20480B	26%	5	21%	4	11%	2	[Gogol]	[Gogol]	[Bulg., Gogol, Turg.]
24576B	26%	5	21%	4	21%	4	[Gogol, Dost.]	[Gogol]	
28672B	26%	5	21%	4	21%	4	[Gogol]	[Gogol]	[Gogol]
32768B	21%	4	26%	5	5%	1		[Bulg., Gogol]	[Bulg., Dost., Gogol]
39321B	26%	5	26%	5	26%	5			[Bulg., Turg.]
45874B	32%	6	16%	3	16%	3	[Turg.]	[Bulg.]	[Bulg., Turg.]
52427B	16%	3	21%	4	21%	4	[Bulg.]	[Bulg., Gogol, Turg.]	[Bulg., Turg.]
58980B	37%	7	16%	3	11%	2	[Bulg., Gogol]	[Bulg.]	[Bulg., Gogol, Turg.]
65533B	16%	3	11%	2	11%	2	[Bulg., Gogol, Turg.]	[Gogol]	[Bulg., Gogol, Turg.]
76455B	16%	3	11%	2	16%	3	[Bulg., Gogol]	[Dost.]	[Bulg., Turg.]
87377B	26%	5	**0%**	**0**	5%	1	perfekt	[Bulg., Gogol]	[Bulg., Gogol, Turg.]
98299B	26%	5	5%	1	5%	1	[Bulg., Tol., Turg.]	[Bulg., Gogol]	[Bulg., Gogol, Turg.]
109221B	42%	8	5%	1	**0%**	**0**	[Bulg., Tol., Turg.]		perfekt
120143B	42%	8	**0%**	**0**	**0%**	**0**	perfekt	[Turg.]	perfekt
131065B	32%	6	16%	3	5%	1	[Bulg., Dost.]		[Bulg., Gogol, Turg.]
149789B	26%	5	**0%**	**0**	5%	1	perfekt	[Bulg.]	[Bulg., Gogol, Turg.]
168512B	16%	3	5%	1	5%	1	[Bulg., Gogol, Turg.]	[Bulg., Gogol]	[Bulg., Gogol, Turg.]
187237B	26%	5	5%	1	**0%**	**0**	[Bulg., Gogol, Turg.]	[Bulg.]	perfekt
205961B	37%	7	5%	1	**0%**	**0**	[Bulg., Gogol, Turg.]		perfekt
224685B	37%	7	**0%**	**0**	**0%**	**0**	perfekt		perfekt
243409B	37%	7	**0%**	**0**	**0%**	**0**	perfekt		perfekt
262133B	53%	10	**0%**	**0**	**0%**	**0**	perfekt	nicht mehr eindeutig	perfekt

Tabelle 3.6: Fehlerquote der Klassifikation. Wir betrachten eine Gruppierung als perfekt, falls jedem der fünf Autoren je ein Unterbaum zugeordnet wird. Befindet sich der Text eines Autors in dem Unterbaum eines anderen Autors, so wird dies als Fehlgruppierung gewertet. Die Anzahl der Fehler geteilt durch die Anzahl der Werke (19) ergibt den angegebenen Prozentsatz. Die Spalte mit der Größenangabe in Byte gibt die Anzahl der gelesenen Bytes des Dokuments an, für das anschließend die Distanzmatrix erstellt wurde. In der Clustering-Spalte stehen die vollständig korrekt erkannten Gruppierungen. Bei einem Fehler sind zwei Gruppen betroffen: bei der einen fehlt ein Dokument, bei der anderen ist ein Dokument zu viel im Unterbaum.

Bei näherer Betrachtung der Versuchsergebnisse fallen einige Besonderheiten auf. So werden zum Beispiel Werke von Turgenev, von Gogol und von Tolstoy sehr früh schon korrekt gruppiert. Wenn wir uns die Tupel zu diesen Arbeiten ansehen, erkennt man ganze Wörter, die referenziert werden, was jedoch nicht bei Werken unterschiedlicher Autoren der Fall ist. Dies legt die Vermutung nahe, dass jeder Autor einen individuellen Wortschatz mit persönlichen Vorlieben hat. Mit zunehmender Textlänge fällt dieser jedoch nicht mehr so stark ins Gewicht, da darüber hinaus auch andere Wörter verwendet werden, was die Wortschätze der Autoren annähert.

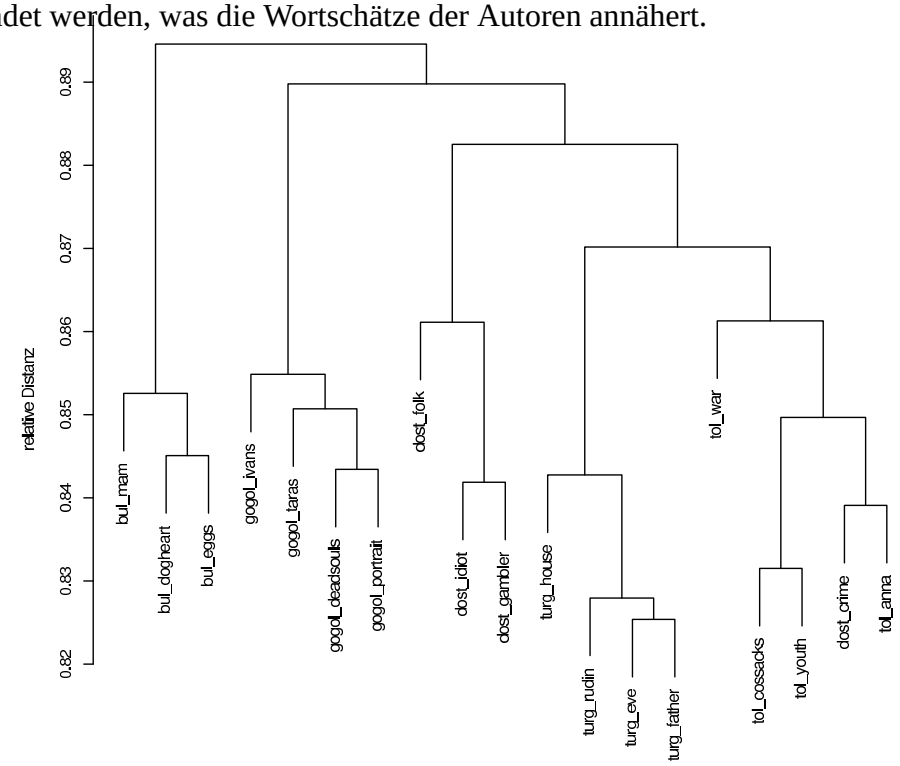


Abbildung 3.12: Das Dendrogramm für die Autorklassifikation der vorverarbeiteten Dokumente mit LZ77 und der Dokumentengröße von 65kB. Bis auf „Schuld und Sühne“ (dost_crime) von Dostojewski, das fälschlicherweise Tolstoy zugeordnet wird, stimmen die Zuordnungen.

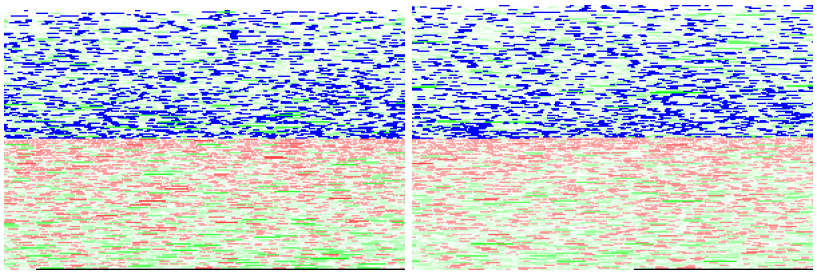
Kompressionsvisualisierung Wie schon zu den Sprachgruppen werden wir auch die Kompression der Autorengruppen visualisieren. Die Schwierigkeit dabei ist jedoch, dass sich die Visualisierung nicht für sehr große Dokumente eignet. Wir werden uns deshalb darauf beschränken, nur die Konkatenation von sehr stark gestutzten Dokumenten (die ersten $65kB$ der Dateien) darzustellen. Wie bereits bei den Sprachen ist auch diesmal der LZ77 Algorithmus so implementiert, dass die gesamte Datei in den Suchpuffer passt. Wir werden hier die last-match Variante darstellen.

Im Folgenden werden wir uns die vier Dokumente: Bulgakov - „Hundeherz“, Bulgakov - „Verhängnissvolle Eier“ , Dostoyevski - „Arme Leute“ und Dostoyevski - „Spieler“ näher ansehen. Diese Dokumente stellten in der Distanzmatrix extreme Werte dar,

und diese werden wir mit Hilfe der Visualisierung untersuchen. Wir werden alle Dokumente mit „Arme Leute“ konkatenieren, dabei bildet die größte Distanz die Konkatenation von „Verhängnissvolle Eier“ und den kleinsten Abstand erhalten wir bei dem Konkatenation mit „Spieler“. Des weiteren haben wir „Hundeherz“ mit „Verhängnissvolle Eier“ und mit „Spieler“ konkateniert um einen Vergleich zu ermöglichen. Die Visualisierungen sind in der Abbildung 3.13 und 3.14 zu sehen. Bei genauem hinsehen erkennt man einen größeren Blauanteil bei den Bildern zu Paaren gleicher Autoren als bei den Bildern mit Paaren unterschiedlicher Autoren. Das deutet darauf hin, dass die Anzahl der Referenzen aus einem Dokument in das andere bei gleichen Autoren (Abbildung 3.13) größer als bei unterschiedlichen Autoren (Abbildung 3.14).

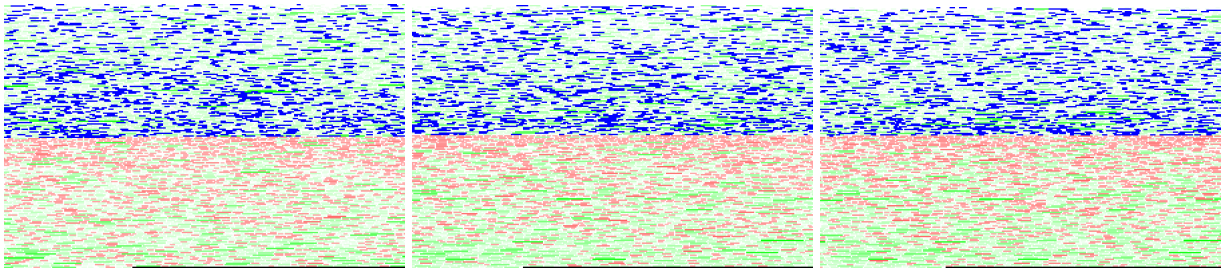
Wenn man sich die Tupelverteilung bei den Fremdreferenzen (Abbildung 3.15), also den Referenzen die auf das linkskonkatenierte Dokument zeigen, ansieht, so ist der Fall nicht mehr ganz so klar wie bei der Sprachgruppierung. Man kann zwar immer noch eine Rechtsverschiebung der Tupelverteilung bei Dokumenten desselben Autors erkennen, diese ist jedoch nicht mehr so klar wie bei den Sprachen. Insgesamt lässt sich sagen, dass die Verteilungen bei den Autoren deutlich homogener ausfallen als bei den Sprachen. Dies liegt zum einen daran, dass die doch sehr klaren Unterschiede zwischen den Zeichenketten, wie dies bei den Sprachen der Fall ist, bei den Autoren nicht mehr gegeben sind. Die Autoren schreiben alle in derselben Sprache, und bis auf persönliche Vorlieben verwenden Sie auch dieselben Wörter. Daher ist auch die Unterscheidung hier schwieriger, was sich auch in den Distanzen niederschlägt.

Auffällig ist auch die Tatsache, dass bei den Autoren grundsätzlich längere Tupel vorkommen. Dies lässt sich so erklären, dass sich aufgrund der gemeinsamen Sprache schneller längere gemeinsame Abschnitte finden lassen. Dies führt, zusammen mit der Länge der Texte, dazu, dass sich während der Kompression immer längere Zeichenketten finden.



(a) bul-doghearts - bul-eggs (b) dost-folk - dost-gambler

Abbildung 3.13: Die Bilder zeigen die Kompression der Konkatenationen von gleichen Autoren.



(a) dost-folk - bul-eggs (b) bul-eggs - dost-gambler (c) bul-dogheart - dost-gambler

Abbildung 3.14: Die Bilder zeigen die Kompression der Konkatenationen von verschiedenen Autoren.

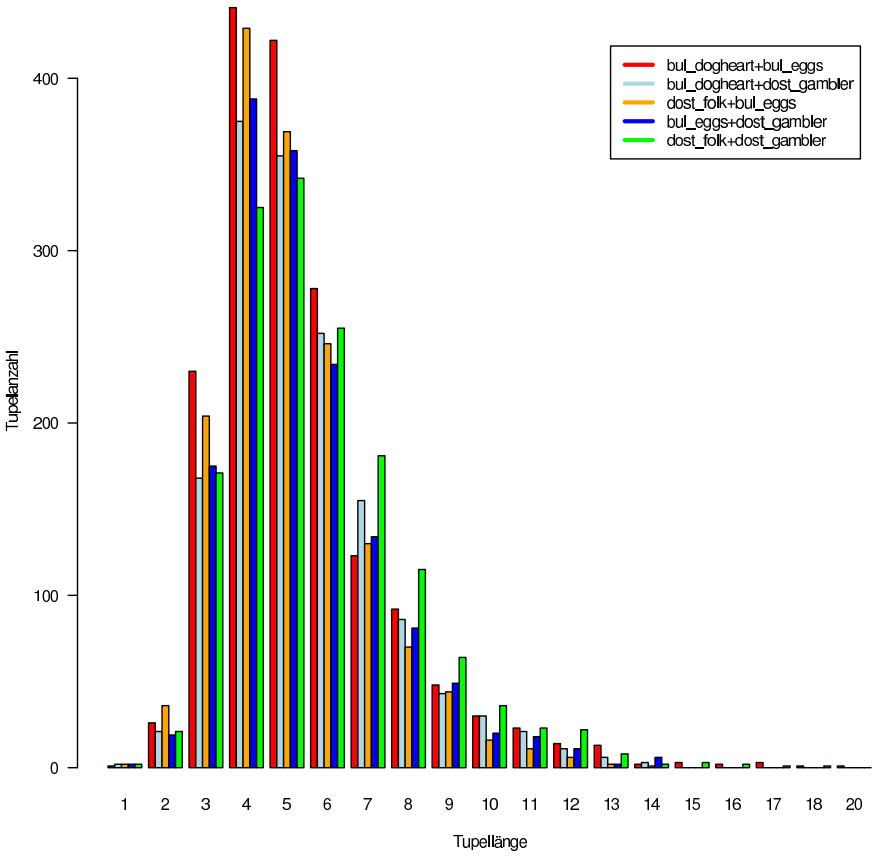


Abbildung 3.15: Fremdtupelverteilung ausgewählter Konkatenationen. Dabei ist wohl am auffälligsten, dass die meisten Tupel eine Länge zwischen 3 und 7 haben. Die Form und die charakteristischen Werte entsprechen in etwa der Wortlängenverteilung (Sigurd et al. 2004) in natürlichsprachlichem Text. Die Parameter der Wortlängenverteilung sind zwar Sprachabhängig, die Verteilung scheint jedoch davon nicht betroffen zu sein.

Fremdtupellänge	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	20
bul-dogheart + bul-eggs	1	26	**230**	**441**	**422**	**278**	123	92	48	30	**23**	14	**13**	2	3	2	3	1	1
dost-folk + dost-gambler	2	21	171	325	342	255	**181**	**115**	**64**	**36**	**23**	**22**	8	2	3	2	1	1	0
bul-dogheart + dost-gambler	2	21	168	375	355	252	155	86	43	30	21	11	6	3	0	0	0	0	0
dost-folk + bul-eggs	2	**36**	204	429	369	246	130	70	44	16	11	6	2	1	0	0	0	0	0
bul-eggs + dost-gambler	2	19	175	388	358	234	134	81	49	20	18	11	2	6	0	0	0	0	0

Tabelle 3.7: Gewichte der Fremdtupel ausgewählter Konkatenationen.

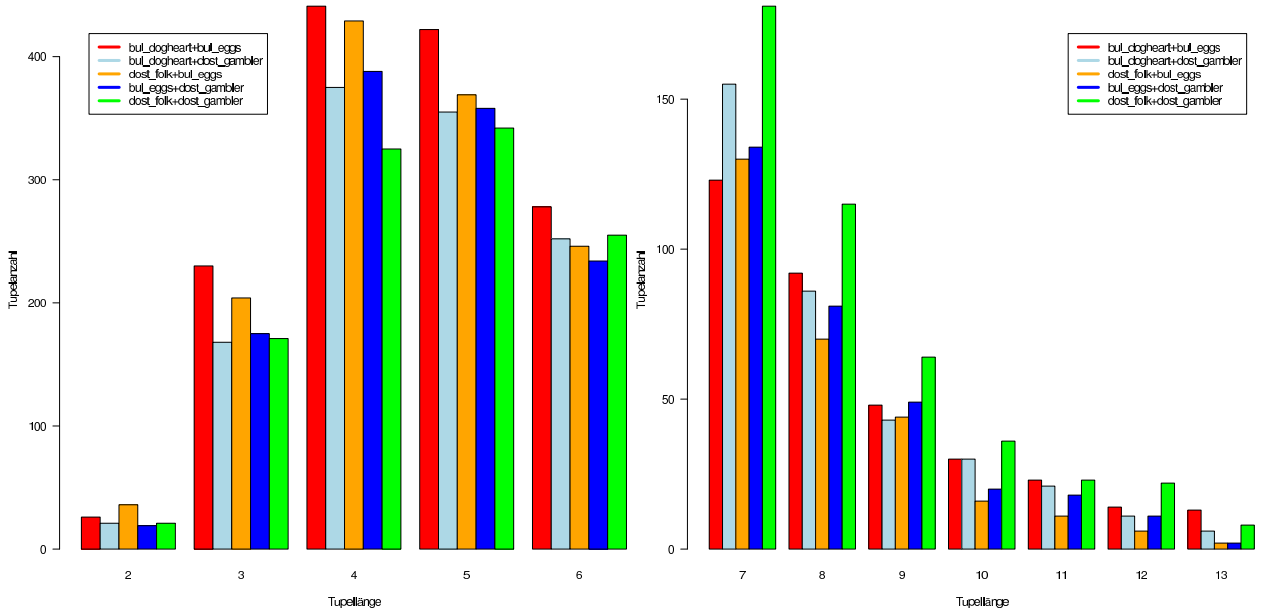


Abbildung 3.16: Fremdtupelverteilung ausgewählter Bereiche der Gesamtverteilung.

Zusammenfassung Die klare Dominanz der gefundenen Wörter im Text bei der Gruppierung von Sprachen lässt sich nicht in dieser Weise bei den Autoren wiederfinden. Zwar gibt es zahlreiche Anzeichen dafür, dass sich auch die Autorentexte durch die gefundenen Wörter unterscheiden, doch lässt sich dies nicht zweifelsfrei aus der Visualisierung und den Statistiken ableiten. Ausgesprochen spannend in diesem Zusammenhang ist die Arbeit von Burkovski et al. (2009), in der zu jedem Text ein Wörterbuch (die Menge der im Text verwendeten Wörter) aufgebaut wird und der Vergleich zweier Texte nur auf der Anzahl der gemeinsamen Wörter im Wörterbuch basiert. Burkovski et al. konnten mit diesem Vorgehen alle bisher gezeigten Ergebnisse exakt rekonstruieren, was die Vermutung verstärkt, dass ausschließlich die im Text enthaltenen Worte für die erfolgreiche Gruppierung verantwortlich sich.

3.3 NCD auf kontinuierlichen Daten

Die Anzahl der Arbeiten zu kompressionsbasierten Ähnlichkeitsmetriken auf kontinuierlichen Daten ist recht übersichtlich. Die dort beschriebenen Ergebnisse entsprechen nicht immer dem, was der Stand der Forschung in diesem Bereich vorgibt. Im Rahmen seiner Studienarbeit hat Thom (2008) die Leistungsfähigkeit der NCD auf kontinuierlichen Daten beleuchtet. Hier wurde in vielen Punkten die Hoffnung auf ein allgemeines Ähnlichkeitsmaß mit herausragenden Fähigkeiten, auch auf kontinuierlichen Daten, zerschlagen. Wir werden uns im Folgenden einige Punkte der Arbeit ansehen, da die Experimente zwar aufschlussreich sind, was die Leistungsfähigkeit angeht, jedoch nur wenig neue Informationen geben, was die Funktionsweise der NCD anbelangt.

3.3.1 NCD auf Grafiken

In seiner Arbeit stellt Thom drei Experimente zur NCD auf kontinuierlichen Daten dar. Das erste betrachtet die Gruppierung von Bildern aus der COIL-100 (Nayar und Murase 1996) Bildbibliothek. Das Zweite behandelt die Gruppierung von vier Personenfarbfotos, wobei der Fokus dieses Versuchs klar auf der qualitativen Evaluierung der Ergebnisse liegt. Das dritte Experiment betrachtet Bilder (die mit einem Zeichenprogramm erstellt worden sind) von geometrischen Figuren. Auch hier ist das primäre Ziel eine qualitative Evaluation. Wir werden uns daher nicht näher mit diesem Experiment beschäftigen, sondern nur die zwei vorangegangenen betrachten.

Coil100

In dem von Thom beschriebenen Versuch geht es darum 150 Coil100 Bilder entsprechend ihrer Gruppenzugehörigkeit zu ordnen. Die Bilder stammen aus den ersten 10 Klassen (die jeweils 15 Bilder enthalten) der Bibliothek. Ziel war es, diese Bilder mit Hilfe der NCD (basierend auf dem *bzip2* Kompressor) zu gruppieren. Dabei erreicht die NCD durchschnittliche Fehlerraten von 12% (Thom 2008, Kapitel 6.4.1). Versuche

mit dem *gzip* Kompressor lieferten durchwegs deutlich schlechtere Ergebnisse (20% Fehlerrate). Neben diesem Experiment selbst ist noch der Vergleich zu einer weiteren relativ simplen Distanzfunktion interessant. Thom betrachtete die Ergebnisse der Gruppierung mit Hilfe eines Histogrammvergleichs. Die dazugehörige Distanzfunktion

$$d_{\text{Hist}}(x,y) = \frac{\sum_{i=0}^{255} |H_i(x) - H_i(y)}{\|x\| + \|y\|}.$$

betrachtet die Differenz der Histogramme $H(\cdot)$ der Bilder x und y. Diese Differenz wird zur besseren Vergleichbarkeit normiert bzgl. der Bildgrößen $\|x\|$ und $\|y\|$. In der Formel steht der Index i für den jeweiligen Grau- oder Farbwert. Eine Gruppierung mit dieser Distanzfunktion liefert zur NCD vergleichbare Ergebnisse.

Personenfotos

Gerade sehr strukturierte und stilisierte Bilder wie sie in der Coil Datenbank vorkommen, können sehr leicht die Interpretation der Ergebnisse erschweren. So ist es zum Beispiel bei den Coil100 Bildern der Fall, dass der Hintergrund nicht immer exakt schwarz ist, sondern von Gruppe zu Gruppe variiert. Um dem entgegen zu wirken, hat Thom (2008) selbst Bilder aufgenommen und den Prozess der Mustererkennung, ähnlich den Text-Kompressionsbildern von Burkovski, visualisiert. Dabei wurden Pixelblöcke, die in dem links-konkatenierten Bild gefunden und aus dem rechtskonkatenierten Bild referenziert wurden, eingefärbt. Damit die Visualisierung nicht zu unübersichtlich wird, wurden nur Tupel ab der Länge 3 eingefärbt. Das Ergebnis kann in Abbildung 3.17 betrachtet werden. Klar zu sehen ist, dass nur sehr wenige lange Tupel erkannt werden. Die meisten Referenzen zeigen auf Bytefolgen der Länge drei und kürzer. In einem Bild lässt sich damit keine semantisch relevante Information extrahieren.

Abbildung 3.17: Visualisierung der Kompression der Personenfotos. Diese Darstellung verdeutlicht, welche Bereiche in einem Bild für die gemeinsame Kompression herangezogen werden. Dabei ist zu sehen, dass die Fremdreferenzen alle in Bereichen liegen, die keine semantische Information tragen.

KAPITEL

4 THEORIE DER KOMPRESSIONSBASIERTEN MUSTERERKENNUNG

Überblick

- Regularisierung und Sparse Coding bilden die Basis für die kompressionsbasierte Mustererkennung. Basis pursuit (Absch. 4.1.2) und Matching pursuit (Absch. 4.1.1) stellen zwei zentrale Verfahren in diesem Kontext dar.
- Redundante Wörterbuchräume sind die Umgebung, in der kompressionsbasierte Verfahren arbeiten. Diese werden hier zu Vektor- (Absch. 4.2) und Hilberträumen erweitert.
- In der kompressionsbasierten Mustererkennung (Absch. 4.6) findet die Theorie der redundanten Wörterbuchräume Anwendung. So beschreibt z.B. Definition 4.5 einen Kompressor und Definition 4.6 stellt die Verbindung zur NCD her.

A theory, in the scientific sense of the word, is an analytic structure designed to explain a set of empirical observations.
Dieses Zitat aus Wikepedia zum Thema *Theory* (Wikipedia 2010d) stellt eine gute Einführung in das aktuelle Kapitel dar. Wir werden im Folgenden eine analytische Struktur entwickeln, die darauf abzielt, bereits bestehende Ergebnisse im Rahmen der kompressionsbasierten Mustererkennung zu erklären, aber auch als Fundament dient, um neues Wissen zu generieren.

In diesem Kapitel werden wir die zugrundeliegenden Methoden einführen. Dieses findet hauptsächlich in Abschnitt 4.1 statt. Dabei spielen die im Rahmen von *Sparse Coding* und *Compressive Sensing* gewonnenen Erkenntnisse eine zentrale Rolle. Wir

werden auf diesen Ideen aufbauen und einen Vektorraum, basierend auf redundanten Wörterbüchern, entwickeln (Abschnitt 4.2). Diese Abschnitte bilden die Basis um bestehende Erkenntnisse, gewonnen im Rahmen der NCD, zu erklären und zu vertiefen (Abschnitt 4.6). Hier werden wir näher auf den Zusammenhang zwischen Funktionen, Dokumenten, Wörterbüchern und Kompression eingehen.

Die später vorgestellten Redundante-Wörterbuchräume bilden die Basis für die Theorie der Kompressionsbasierten Mustererkennung. Sie bilden, wie wir in Abschnitt 4.5 sehen werden, jedoch auch das Fundament für eine noch weitergehende Idee. Der allgemeinen Behandlung nichtvektorieller Daten innerhalb eines Vektorraums.

Weitere Überlegungen zu den Themenfeldern Kompression und Mustererkennung finden sich in den Arbeiten Klenk und Heidemann 2008 und Klenk und Heidemann 2009.

Nomenklatur und Notation

Im Rahmen dieser Arbeit beschäftigen wir uns hauptsächlich mit reellwertigen Funktionen $f, g \in \mathcal{H} \subset L^2(\mathbf{K})$ definiert auf einer kompakten Teilmenge der reellen Achse $\mathbf{K} \subset \mathbf{R}$. Diese sind von endlicher Norm in einem Hilbertraum $\mathcal{H}$

$$\|f\|_2 = \left(\int |f(t)|^2 dt \right)^{1/2} < \infty.$$

In diesem Raum sei das innere Produkt $\langle f, g \rangle$ gegeben als Integral über das Produkt der beiden Funktionen

$$\langle f, g \rangle = \int f(t) \cdot g(t) dt.$$

Des weiteren werden wir uns mit sogenannten Wörterbüchern (oft auch mit dem englischen Begriff *Dictionaries* bezeichnet) $\mathcal{D} = \{\phi_i\}_{i \in \mathbf{I}} \subset \mathcal{H}$ als einer Teilmenge, bestehend aus abzählbar vielen Elementen eines Hilbertraums über einer Indexmenge $\mathbf{I}$, beschäftigen. Die Funktionen $\phi \in \mathcal{D}$ haben allesamt die Norm $\|\phi_i\|_2 = 1$. Wenn wir den vom Wörterbuch aufgespannten Raum betrachten, so immer nur bezüglich Koeffizienten aus ℓ^1 oder, je nach Kontext auch ℓ^2, also

$$\mathcal{H} = \left\{ v | v = \sum_{i \in \mathbf{I}} \alpha_i \phi_i \,\text{mit}\, \phi_i \in \mathcal{D}, \alpha \in \ell^{\{1|2\}} \right\}.$$

Das Besondere an einem Wörterbuch, im Gegensatz zu einer Basis ist, dass es üblicherweise hochgradig redundant (bezüglich des eigenen Aufspanns) ist. Das bedeutet, dass es sich dabei eben nicht um eine linear unabhängige Teilmenge handelt. Generell ist es sogar wünschenswert, besonders viele Aspekte einer Gruppe von Funktionen möglichst genau abzudecken und dabei Redundanzen in Form linearer Abhängigkeiten in Kauf zu nehmen. Abgesehen davon werden wir Funktionen nicht nur in Bezug auf ihre Funktionenraumrepräsentation $f, g \in \mathcal{H}$ betrachten, sondern uns auch mit deren Sequenzraumrepräsentation $x_f, x_g \in \mathcal{G}$ beschäftigen. Dort bilden die einzelnen

Komponenten x_f^i die Koeffizienten der linearen Superposition der Wörterbuchelemente bezüglich der Funktion f.

$$f = \sum_{i \in \mathbf{I}} x_f^i \phi_i$$

Die Menge der Sequenzraumrepresentation $\{x_f\}_{f \in \mathcal{H}} \subseteq \mathcal{G}$ bildet ebenfalls einen Hilbertraum, wobei die Elemente x_f nur abzählbar viele Komponenten x_f^i mit Indizes aus der Indexmenge $\mathbf{I}$ enthalten.

In beiden Räumen, dem Hilbertraum und dem korrespondierenden Sequenzraum, werden wir mit den entsprechenden Normen L^p und ℓ^p, definiert als

$$\|f\|_p = \left(\int |f(t)|^p dt \right)^{1/p} \qquad \text{und} \qquad \|\alpha\|_p = \left(\sum_{i \in \mathbf{I}} |\alpha_i|^p \right)^{1/p}$$

arbeiten. In diesem Zusammenhang ist ℓ^0 auf dem Sequenzraum definiert als Grenzwert der Folge $\|x\|_0 = \lim_{p \to 0} \|x\|_p < \infty$ wobei $\|x\|_0$ die Anzahl der nicht-Null Einträge in x ist.

Ein besonderer Aspekt dieser Arbeit – dem wir uns im Laufe dieses Kapitels nähern werden – ist, dass wir uns von der Bedingung, die Wörterbuchelemente müssten aus einem Hilbertraum entstammen, lösen können. Dies erlaubt uns die Behandlung allgemeiner, also auch nichtvektorieller Daten, innerhalb eines Vektorraums. Wir werden dazu den Hilbertraum $\mathcal{H}$ durch einen Datenraum $\mathcal{X}$ ersetzen.

Des Weiteren werden wir mit den zwei Projektionsoperatoren P_V und P_W sowie P_{V_m} und P_{W_m} arbeiten. Diese bilden den Hilbertraum $\mathcal{H}$ auf den abgeschlossenen linearen Aufspann $V \subseteq \mathcal{H}$ des Wörterbuchs oder dessen orthogonalem Komplement W ab. Dabei deutet der Index m an, dass nur m, oder weniger Elemente den Aufspann bilden, also für jedes $x \in P_{V_m}$ gilt $\|\alpha\|_0 \leq m$ für $x = \sum_i \alpha_i \phi_i$.

Kompressionsbasierte Verfahren

Das Hauptaugenmerk dieses Abschnitts liegt in der Entwicklung einer Theorie, die diskrete und kontinuierliche kompressionsbasierte Mustererkennung vereinigt. Das erfordert, dass die Notation, sowohl in Bezug auf die Daten, als auch auf die Algorithmen keine Bruchstellen bildet. Um im Rahmen der gängigen Bezeichnungen zu bleiben, werden wir uns hauptsächlich über *Funktionen* unterhalten, ggf. jedoch auch die Bezeichnung *Dokumente* akzeptieren. Dies erlaubt eine schlüssigere Nomenklatur und ist im mathematischen Sinne korrekt, da Funktionen üblicherweise als Relation zwischen Mengen definiert sind (siehe Simovici et al. 2008). Der in diesem Kontext verwendete Funktionenbegriff ist identisch mit dem oben eingeführten, wobei wir für diskrete Funktionen auch eine kompakte Teilmenge einer Indexmenge $\mathbf{K_I} \subset \mathbf{I}$ zulassen und Funktionen- sowie Sequenzraumrepräsentationen synonym verwenden.

4.1 Regularisierung und Sparse Coding

Regularisierung (Tichonov und Arsenin 1977) ist seit der Popularität von Support Vector Machines (Vapnik 1998; Schölkopf und Smola 2002) eine feste Größe in der Pattern Recognition Community. Die Idee, sogenannte ill-posed Problems, also nicht wohldefinierte Fragestellungen eindeutig mit Hilfe gängiger Optimierungsmethoden zu lösen, hatte schon zu Tichonovs Zeiten seinen Reiz. Doch gerade im Rahmen maschineller Lernverfahren – dort ist fast keine Fragestellung wohldefiniert, im klassischen Sinne – fanden sich zahlreiche Anwendungsbereiche.

Zuerst jedoch die Frage, was wohldefiniert ist: Eine Problemstellung F mit gegebenem Datum u aus einem metrischen Raum U, charakterisiert durch ihre Lösungsmenge $Fu = V' \subset V$ eines ebenfalls metrischen Raums V ist wohldefiniert (nach Tichonov und Arsenin 1977), falls folgende Bedingungen erfüllt sind:

1. zu jedem $u \in U$ ist existiert eine Lösung $Fu \in V$,
2. die Lösung Fu ist eindeutig und
3. das Problem Fu ist stabil bezüglich kleiner Änderungen von u.

Bedingungen (1) und (2) sind der mathematischen Wohldefiniertheit geschuldet. Bedingung (3) ist ein Tribut an die physikalischen Wurzeln der Idee und auch an die Lösbarkeit mit numerischen Verfahren. Gerade die mathematische Wohldefiniertheit (Existenz und Eindeutigkeit) wird uns später noch intensiv beschäftigen.

Tichonov und Arsenin sowie Vapnik bauen auf dieser Idee auf, um sogenannte inverse Probleme zu lösen. Bei diesen Problemstellungen versucht man, gegeben zwei Mengen U und V und eine funktionale Abhängigkeit

$$Fv = u$$

zwischen diesen zu bestimmen, wobei F und u bekannt sind. Die Lösungsmenge, also das inverse Bild $F^{-1}[\{u\}]$, enthält nicht zwangsläufig nur ein Element, was es zu einer nicht wohldefinierten Problemstellung macht. Der Ansatz zur Lösung baut darauf auf, eine weitere Bedingung für die Lösung anzugeben, mit deren Hilfe das Problem eindeutig lösbar wird. Etwas strenger ausgedrückt bedeutet dies, sich ein v' aus der Menge aller möglicher Lösungen $\{v \in V | Fv = u\} = V'$ auszuwählen, das bezüglich eines Stabilitätskriteriums S minimal ist. Das Funktional hat dann die folgende Form

$$T_u(v) = ||Fv - u||^2 + \alpha S(v). \tag{4.1}$$

Damit wir T_u minimieren können, setzen wir die Kompaktheit der Lösungsmenge V' voraus. Des weiteren fordern wir, dass die Abbildung F kontinuierlich ist und S ein konvexes Funktional ist. Damit können wir die Existenz und die Eindeutigkeit einer Lösung garantieren (siehe Vapnik 1998, Appendix to Chapter 1).

In den nun folgenden Abschnitten werden wir uns einer Weiterentwicklung dieser grundsätzlichen Idee zuwenden, dem sogenannten Sparse Coding. Dabei ist es das

Ziel, eine Funktion als Superposition von Elementen eines hochgradig redundanten Wörterbuchs darzustellen. Den Zusammenhang erhält man, wenn man sich vor Augen führt, dass die Darstellung einer Funktion als Superposition redundanter Funktionen

$$f = \sum_{i \in \mathbf{I}} \alpha_i \phi_i \quad \text{oder in Matrixschreibweise} \quad f = \Phi\alpha$$

eben nicht eindeutig ist. Das inverse Bild $\Phi^{-1}[\{f\}]$ weist einer Funktion f einen Untervektorraum zu. Da alle Elemente diese Raums gleichberechtigte Lösungen des inversen Problems sind, braucht man eine Möglichkeit, ein priveligiertes Element auszuwählen. Tichonov und Arsenin nutzten hierzu das eher allgemeine Stabilitätskriterium. Wir werden im weiteren nach einer möglichst „dünnen“ Lösung – also einer Lösung mit möglichst wenigen nicht-null Einträgen – suchen.

4.1.1 Matching pursuit

Eine Lösung für das inverse Problem zu finden, die dünn besetzt ist – also möglichst wenige von Null verschiedene Einträge besitzt – bedeutet die Kombination an Wörterbuchelementen zu finden, die mit möglichst wenig Elementen die gesuchten Funktion möglichst gut approximiert. Das Finden einer exakten Lösung ist nach Davis et al. (1997) NP-hart, für alle realen Fragestellungen also im allgemeinen nicht lösbar. Wir werden uns deshalb hier mit zwei approximativen Ansätzen begnügen. Der erste Weg, beschrieben von Mallat und Zhang (1993), ist ein sogenanntes Greedy-Verfahren zur Bestimmung einer möglichst dünnen Repräsentation. Bei jedem Iterationsschritt wird dabei versucht, den Approximationsfehler durch Hinzunehmen eines weiteren Wörterbuchelements zu minimieren. Solch ein Vorgehen garantiert jedoch kein globales Optimum, also keine echt minimale Repräsentation. Der klare Vorteil dieses Verfahrens liegt in seiner Einfachkeit und der hohen Geschwindigkeit.

In einem Hilbertraum $\mathcal{H}$, gegeben durch den Aufspann des Wörterbuchs

$$\mathcal{H} = \left\{ v | v = \sum_{i \in \mathbf{I}} \alpha_i \phi_i \,\text{mit}\, \phi_i \in \mathcal{D}, \alpha \in \ell^2 \right\},$$

sieht das Verfahren wie folgt aus. Ein Wörterbuch $\mathcal{D} = \{\phi_i\}_{i \in \mathbf{I}}$ mit Indexmenge $\mathbf{I}$ sei gegeben, wobei die Einträge alle normiert sein müssen $\|\phi_i\|_2 = 1$. Das Wörterbuch wird dabei so gewählt, dass es den ganzen Raum aufspannt, also eine Basis des Raums enthält. Um nun eine Linearkombination von Wörterbuchelementen zu der Funktion f zu finden, haben Mallat und Zhang das Matching Pursuit Verfahren entwickelt, das iterativ den Wörterbucheintrag auswählt, der das Residuum

$$Rf = f - \langle f, \phi_i \rangle \phi_i$$

minimiert. Das führt zu einem Algorithmus, bei dem, ausgehende von einem Wörterbucheintrag ϕ_i, die Funktion auf den Raum der verbleibenden Wörterbucheinträge projiziert wird

$$R^n f = \langle R^n f, \phi_{i_n} \rangle \phi_{i_n} + R^{n+1} f.$$

Ausgehend davon wird ein neuer Wörterbucheintrag ausgewählt. In jedem Schritt wird also ein weiteres ϕ_i mit

$$\phi_{i_n} = \arg\max_{\phi_j \in \mathcal{D}} |\langle R^n f, \phi_j \rangle|$$

ausgewählt, das nach m Iterationen zu folgender Repräsentation führt

$$f = \sum_{n=0}^{m} \langle R^n f, \phi_{i_n} \rangle \phi_{i_n} + R^{m+1} f.$$

Ein ganz besonders interessanter und vielversprechender Aspekt dieser Approximation den wir jedoch im Rahmen dieser Arbeit nicht weiter verfolgen können ist das *Coherent Matching Pursuit Denoising* (Mallat 2009; Mallat und Zhang 1993, page 655). Dabei geht es darum, ähnlich der menschlichen Wahrnehmung, nur die Aspekte einer Funktion darzustellen, die in einem vorher definierten Kontext von Interesse sind. So ist der Mensch ohne Weiteres in der Lage, aus einer Gräuschkulisse für ihn interessante Geräusche herauszufiltern. Diesen Effekt kann man auch mit Hilfe von Sparse Coding erzielen, indem man die interessanten Geräusche als Elemente eines von einem Wörterbuch aufgespannten Vektorraums anssieht. Verfahren wie Compressive Sensings behandelt diese Thematik ausgiebig (Donoho 2006; Candès und Romberg 2006; Candès et al. 2006; Haupt und Nowak 2006)[1].

Fast Sparse Coding

Geschwindigkeit ist üblicherweise ein großer Vorteil sogenannter Greedy-Verfahren. Für das Matching Pursuit Verfahren gibt es noch eine *Fast Network Calculations* Variante, die rekursiv die Projektion des Residuums aktualisiert und sich dadurch die Neuberechnung des Residuums vermeidet.
Hier der Algorithmus wie von Mallat (2009, Seite 645) beschrieben:

1. *Initialisierung.* Setze $m = 0$.
2. Berechne $\{\langle f, \phi_i \rangle\}_{i \in \mathbf{I}}$ in $\mathcal{D}$.
3. *Best Match.* Finde $\phi_{i_m} \in \mathcal{D}$, sodass
$$|\langle R^m f, \phi_{i_m} \rangle| = \max_{i \in \mathbf{I}} |\langle R^m f, \phi_i \rangle|$$
4. *Update.* Für jedes $\phi_i \in \mathcal{D}$ mit $\langle \phi_{i_m}, \phi_i \rangle \neq 0$,
$$\langle R^{m+1} f, \phi_i \rangle = \langle R^m f, \phi_i \rangle - \langle R^m f, \phi_{i_m} \rangle \langle \phi_{i_m} \phi_i \rangle. \tag{4.2}$$

[1] Weitere Referenzen und eine ausgiebige Diskussion der Themen Compressive Sensing und Sparse Coding können auf der Homepage der Digital Signal Processing (DSP) Grupper der Rice University `http://dsp.rice.edu/cs` gefunden werden

5. Abbruch falls $\|R^{m+1}f\|$ klein genug, sonst weiter bei Schritt 3.

Eine weitere Möglichkeit, den Berechnungsaufwand zu reduzieren, ist das letzte innere Produkt (4.2) einmal zu berechnen und abzuspeichern. Da es sich im Laufe der Iterationen nicht ändert, kann man so ebenfalls Rechenzeit sparen.

Konvergenz

In diesem Abschnitt werden wir uns mit der Konvergenz des Matching Pursuit Verfahrens beschäftigen. Da dieses Thema auch später noch von Relevanz sein wird, werden wir uns auch die dazugehörigen Beweise ansehen. Eine ausführliche Diskussion des Themas findet sich in den Arbeiten von Mallat und Zhang (1993) und Mallat (2009).
Satz 4.1 (Decay of Matching Pursuit – Mallat und Zhang 1993)**.**
Sei $f \in \mathcal{H} = \operatorname{span}\mathcal{D}$. Für jedes $m > 0$ und $\alpha \in (0, 1]$, gilt für das Residuum $R^m f$

$$\|R^m f\|^2 \leq (1 - \alpha^2 \mu_{min}(\mathcal{D})^2)^m ||f||^2 \qquad \text{with} \qquad 1 \geq \mu_{min}(\mathcal{D}) > 0.$$

Dabei ist

$$\mu_{min}(\mathcal{D}) = \inf_{f \in \mathcal{H}, f \neq 0} \sup_{i \in \mathbf{I}} \frac{|\langle f, \phi_i \rangle|}{||f||}.$$

Der Beweis dieses Satzes (der übrigens in seiner ursprünglichen Form nur für endlichdimensionale Hilberträume $\mathcal{H}$ gilt) folgt dem Weg beschrieben in Mallat (2009). Da wir für $\mathcal{H}$ den Aufspann des Wörterbuchs annehmen, können wir den Beweis etwas abkürzen.

Beweis. Das Wörterbuchelement ϕ_{i_m}, das im m-ten Iterationsschritt ausgewählt wird, gehorcht $|\langle R^m f, \phi_{i_m} \rangle| \geq \alpha \sup_{i \in \mathbf{I}} |\langle R^m f, \phi_i \rangle|$. Da $R^{m+1}f$ und ϕ_{i_m} orthogonal zu einander stehen – das Residuum $R^{m+1}f$ erhält man ja erst dadurch, dass man ϕ_{i_m} davon abzieht – gilt für die Energie der Funktion

$$\|R^m f\|^2 = |\langle R^m f, \phi_{i_m} \rangle|^2 + \|R^{m+1} f\|^2.$$

Daraus folgt wiederum

$$\frac{\|R^{m+1} f\|^2}{\|R^m f\|^2} = 1 - \left| \left\langle \frac{R^m f}{\|R^m f\|}, \phi_{i_m} \right\rangle \right|^2 \leq 1 - \mu(R^m f, \mathcal{D})^2$$

wobei

$$\mu(R^m f, \mathcal{D}) = \sup_{i \in \mathbf{I}} \left| \left\langle \frac{R^m f}{\|R^m f\|}, \phi_i \right\rangle \right| \leq 1.$$

Zusammen mit der eingangs erwähnten Beziehung zwischen dem ausgewählten Wörterbuchelement und dem bestmöglichen Wörterbuchelement ergibt dies

$$\frac{\|R^{m+1} f\|^2}{\|R^m f\|^2} \leq 1 - \alpha^2 \mu_{\min}(\mathcal{D})^2.$$

Wenn man diese Ungleichung wiederholt anwendet, erhält man

$$\|R^m f\|^2 \leq (1 - \alpha^2 \mu_{\min}(\mathcal{D})^2)^m \|f\|^2.$$

Dies konvergiert gegen 0 falls $\mu_{\min}(\mathcal{D})$ größer als 0 ist.

Abkürzung: Damit $\mu_{\min}(\mathcal{D}) = 0$ wird, muss ein $f' \neq 0$ existieren für das $\langle f', \phi_i \rangle = 0$ für alle $i \in \mathbf{I}$ gilt. Was aber auch bedeutet, dass f' orthogonal zum Aufspann von $\mathcal{D}$ steht. Dies wiederspricht der Annahme, dass f' ein Element des Aufspanns ist.

Originalweg: Hier betrachten wir $\mathcal{H} = \mathbf{R}^n$ und $|\mathcal{D}| = m$ mit $n < m$ wobei $\mathcal{D}$ vollständig ist. Um nun zu zeigen, dass $\mu_{\min}(\mathcal{D}) > 0$ ist, betrachten wir eine Folge $(f_i)_{i \in \mathbf{N}}$ von, $\|f_i\| = 1$ normierten Funktionen für die

$$\lim_{i \to \infty} \sup_{j \in \mathbf{I}} |\langle f_i, \phi_j \rangle| = 0$$

gilt. Da nun die Einheitskugel in $\mathbf{R}^n$ kompakt ist, existiert eine Teilfolge f_{i_k} die gegen eine Funktion mit Einheitsnorm konvergiert. Daraus folgt

$$\sup_{j \in \mathbf{I}} |\langle f, \phi_j \rangle| = \lim_{k \to \infty} \sup_{j \in \mathbf{I}} |\langle f_{i_k}, \phi_j \rangle| = 0$$

und somit auch $\langle f, \phi_j \rangle = 0$ für alle $j \in \mathbf{I}$. Da $\mathcal{D}$ vollständig ist und damit eine Basis enthält, bedeutet dies zwangsläufig $f = 0$ im Gegensatz zur oben gestellten Annahme. □

Somit haben wir gezeigt, dass das Residuum mit jedem Schritt abnimmt und die Approximation besser wird. Das Matching Pursuit Verfahren ist jedoch nicht in der Lage, ein globales Optimum bezüglich der Approximationsgüte und der Anzahl der benötigten Wörterbuchelemente zu garantieren. Es lassen sich sogar sehr einfache Fälle konstruieren in denen diese Verfahren versagen, wie Mallat (2009, Seite 659) zeigt. Einen möglichen Ausweg bietet die Basis Pursuit Methode, die wir uns im Folgenden ansehen werden.

4.1.2 Basis pursuit

Das von Chen et al. (1996) entwickelte Verfahren zur Berechnung dünner Repräsentationen x für eine Funktion f basiert auf der Idee, dass sich die Problemstellung

$$P0 \qquad \min \|x\|_0 \quad \text{mit} \quad f = \sum_{i \in \mathbf{I}} x_i \phi_i$$

als lineares Optimierungsproblem

$$P1 \qquad \min \|x\|_1 \quad \text{mit} \quad f = \mathcal{D}x$$

darstellen, wobei wir das Wörterbuch in naheliegender Weise als linearen Operator sehen, lässt (für weitere Informationen über lineare Optimierung bieten sich folgende

Monographien an: Nocedal und Wright 2000; Luenberger 1969). Von besonderer Bedeutung ist dabei die ℓ^1-Norm. Diese ersetzt die ℓ^0-Norm als Minimierungskriterium. Das Problem $P0$ ist, wie schon eingangs erwähnt, NP-hart und damit für interessante Problemstellungen nicht exakt lösbar. $P1$ hingegen lässt sich mit Hilfe moderner Verfahren, wie zum Beispiel den sogenannten Inner-Point Methoden, auch für sehr große Problemstellungen lösen. Deren Verfügbarkeit und die Erkenntnis, dass eine Minimierung der ℓ^1 Norm, unter gewissen Umständen, auch gleichzeitig die ℓ^0 Norm minimiert, führte zur Entwicklung des Basis Pursuit Verfahrens.

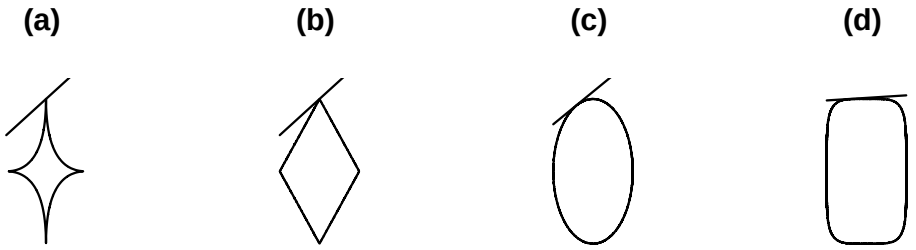


Abbildung 4.1: Einheitskugeln B_p für verschiedenen ℓ^p-Normen: (a) $p = 1/2$, (b) $p = 1$, (c) $p = 2$, (d) $p = 5$. Die anliegende Gerade verdeutlicht die Minimierung bezüglich der jeweiligen Norm. Deutlich zu sehen ist, wie bei $p = 1$ die Zahl der möglichen Minima maximal wird und wie für $p < 1$ das Problem nicht mehr konvex ist.

Um ein besseres Verständnis für das Kriterium der Dünnbesetztheit zu erlangen, schauen wir uns an, was passiert, wenn wir zu einem Vektor $v \in \mathbf{R}^n$ eine dünne Repräsentation $x \in \mathbf{R}^m$ suchen. Das gesuchte x liegt dann auf einer Hyperebene

$$H = \{x \in \mathbf{R}^m | \mathcal{D}x = v\} = x_0 + \mathrm{kern}(\mathcal{D})$$

aufgespannt durch den Untervektorraum des Kerns $\mathrm{kern}(\mathcal{D})$ der linearen Abbildung $\mathcal{D}$ und verschoben um eine Lösung $x_0 \in \mathcal{D}^{-1}[\{v\}]$ des inversen Problems. Ziel ist es, aus H das Element mit der geringsten Norm zu finden. Dies bedeutet, man bläst den ℓ^p-Ball

$$B_p = \{x \in \mathbf{R}^m | \|x\|_p \leq \epsilon\}$$

solange auf, bis dieser sich mit der Hyperebene H schneidet. Auf dieser Schnittfläche liegt dann die minimale Lösung. Für $p > 1$ besteht die Schnittfläche nur aus einem Punkt, für $p = 1$ jedoch können mehrere Lösungen existieren (dies liegt daran dass ℓ^p für $p > 1$ strikt konvex, für $p = 1$ jedoch nur konvex ist). Abbildung 4.1 verdeutlicht diesen Sachverhalt noch einmal visuell, wodurch zu sehen ist, dass B_p für $p \leq 1$ deutlich exponierter nahe den Achsen ist als dies für $p > 1$ der Fall ist. Für $p > 2$

nimmt sogar der Flächeninhalt deutlich zu und begünstigt die großflächigen Bereiche zwischen den Achsen. Für die oben genannten Optimierungsprobleme bedeutet dies, dass für $p \leq 1$ die Wahrscheinlichkeit, eine minimale Lösung auf den Achsen zu finden, größer ist als für $p = 2$. Obwohl ℓ^1 nicht strikt konvex ist (Papadopoulos 2005, Kapitel 7.3) ist zwar eine numerische Optimirierung möglich, aber diese liefert nicht zwangsläufig ein eindeutiges Ergebnis. Die Frage ist nun, wann das Problem $P1$ ein eindeutiges Minimum besitzt. Hier hilft uns folgende Definition weiter:

Definition 4.1 (Charakteristische Zahl Gribonval und Nielsen (2007))**.** *Die charakteristische Zahl $m(\mathcal{D}) \in \mathbf{N}$ eines Wörterbuches $\mathcal{D}$ ist das Supremum aller natürlicher Zahlen m mit der Eigenschaft, dass für alle $f \in \operatorname{span}\mathcal{D}$ die korrespondierende Sequenzraumrepräsentation*

$$f = \sum_{i \in \mathbf{I}} x_i \phi_i \quad \textit{mit} \quad \|x\|_0 \leq m$$

eindeutig ist.

Satz 4.2 (ℓ^p-dünnste Repräsentation Gribonval und Nielsen (2007))**.** *Sei $\mathcal{D}$ ein Wörterbuch eines separierbaren Hilbertraums und $m \leq m(\mathcal{D})$ eine natürliche Zahl, so gilt für alle $x \in \mathcal{G}$ mit*

$$\|x\|_0 \leq m \quad \textit{und} \quad f = \sum_{i \in \mathbf{I}} x_i \phi_i,$$

dass x die (eindeutige) dünnst mögliche Repräsentation von f ist.

Der Beweis zu Satz 4.2, neben weiteren interessanten Aspekten des Sparse Codings in Banachräumen, findet sich in der Arbeit von Gribonval und Nielsen (2007).

4.2 Vektorräume basierend auf redundanten Wörterbüchern

In diesem Abschnitt werden wir uns mit der Frage beschäftigen, wie ein Vektorraum aussieht, der nicht von einer Basis, sondern von einem redundanten Wörterbuch aufgespannt wird. Von zentraler Bedeutung ist dabei der Aspekt der Redundanz des Wörterbuchs. Sie führt nämlich dazu, dass wir keine eindeutige Koordinatenrepräsentation eines Vektors[2] mehr erhalten, sondern Mengen von Koordinatenvektoren, die den selben Vektor repräsentieren. Ein Vektorraum basierend auf redundanten Wörterbüchern (oder engl. Redundant Dictionary Space, RDS) $\mathcal{R}$ ist daher die Äquivalenzklasse eines Vektorraums, was wieder einen Vektorraum ergibt (Rudin 1991).

[2]Weitere Informationen über den Zusammenhang zwischen Vektoren und deren Koordinaten findet man in dem Monograph von (Conway 1990, Chapter 1 4)

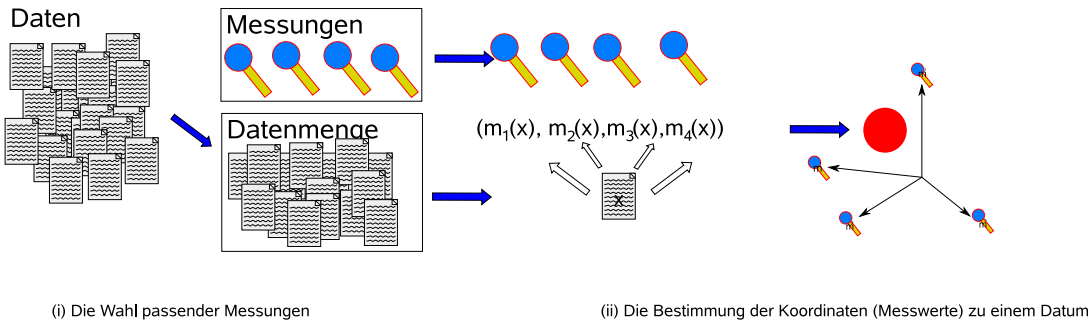


Abbildung 4.2: Die zwei Schritte (Auswahl der Messungen und Abbildung auf die Koordinaten) des gewöhnlichen Koordinatenabbildungsprozesses. Der rote Punkt in dem Koordinatensystem verdeutlicht den Fehler, der durch die Korrelation der Messungen und damit auch der Koordinaten eingeführt wird. Aufgrund dieser fehlerhaften Darstellung wird eine exakte Positionsangabe für ein Datum erschwert.

4.3 Der gewöhnliche Koordinatenabbildungsprozess

Nichtvektorielle Daten werden normalerweise als eine Folge von Messungen behandelt (Abbildung 4.2 zeigt den Prozess der Abbildung nichtvektorieller auf vektorielle Daten). Diese Messungen sind, normalerweise, hochgradig redundant. Text, zum Beispiel, wird gemeinhin als ein Vektor von Termhäufigkeiten repräsentiert. Dabei gibt der Wert jeder Koordinate die Häufigkeit an, mit der der korrespondierende Term in dem betrachteten Text vorkommt. Hier gilt es zu beachten, dass:

1. Die Dimensionalität der Termhäufigkeitsvektoren sehr groß ist, 40.000 Dimensionen und mehr sind eher die Regel als die Ausnahme.

2. Die meisten Koordinaten eines Termhäufigkeitsvektors haben den Wert 0, die Vektoren sind daher sehr dünn besetzt.

3. Die einzelnen Koordinaten sind stark korreliert, das heißt, dass das Auftreten bestimmter Worte, also Koordinaten mit Werten ungleich null, dazu führt, dass andere Worte ebenfalls auftreten.

Die Korrelation der Koordinaten führt auch zu fehlerhaften Vektorraumoperationen. Gerade bei der Norm oder dem Skalarprodukt tritt dies besonders deutlich zutage. Um ein Gefühl für den Einfluss zu bekommen, den eine solche Repräsentation hat, werden wir eine Abschätzung des Fehlers bestimmen. Dazu gehen wir davon aus, dass es einen Datenvektorraum $\mathcal{X} \subseteq \mathbf{R}^n$ gibt und dessen orthonormale Basis bekannt ist. Dieser Raum wird im Folgenden als Referenz für unsere Berechnungen dienen. Des Weiteren gehen wir davon aus, dass die Messungen (μ^i) mit $i \in \{1 \ldots m\}$ ebenfalls aus dem Datenraum $\mathcal{X}$ stammen und, dass wir mindestens soviel Messungen wie Dimensionen im Datenraum haben. Die Messungen sollen ja alle Informationen des Datenraums enthalten. Das heisst: wenn wir n Dimensionen im Datenraum haben,

sollten wir m Messungen mit $m \geq n$ haben. Damit die Messungen auch wirklich alle Informationen des Ursprungsraums wiedergeben nehmen wir des Weiteren an, dass sie nicht negativ sind und folgendes gilt:

$$\mu_i^j \geq 0 \quad \text{und} \quad \mu_i^i = 1 \quad \text{für } i \leq n \text{ und } j \leq m.$$

Auf diese Weise erhalten wir den Wert einer Messung $\mu^i(x)$ zu einem gegebenen Datum x durch Berechnung des Skalarprodukts

$$\mu^i(x) = \langle \mu_i, x \rangle.$$

Die vektorielle Repräsentation des Datums x ist somit gegeben durch

$$v = (\mu^1(x), \ldots, \mu^m(x)) = (\langle \mu^1, x \rangle, \ldots, \langle \mu^m, x \rangle).$$

Wenn wir zwei Datenelemente x^1, x^2 und deren korrespondierende Vektorraumrepräsentationen v^1, v^2 ansehen, so können wir erkennen, dass der Abstand zwischen den zwei Vektorraumrepräsentationen sich von dem Abstand der Datenraumelemente unterscheidet. Dieser Unterschied lässt sich, wie wir nun sehen werden, als additiver Term darstellen.

$$\begin{aligned}
\|v^1 - v^2\| - \|x^1 - x^2\| &= \sum_{i=1}^{m} (v_i^1 - v_i^2)^2 - \|x^1 - x^2\| \\
&= \sum_{i=1}^{m} (\sum_{j=1}^{n} \mu_j^i (x_j^1 - x_j^2))^2 - \|x^1 - x^2\| \\
&\geq \sum_{i=1}^{n} ((x_i^1 - x_i^2) + \sum_{j \neq i}^{n} \mu_j^i (x_j^1 - x_j^2))^2 - \|x^1 - x^2\| \\
&\geq \sum_{i=1}^{n} ((x_i^1 - x_i^2)^2 + 2 \cdot (x_i^1 - x_i^2) \sum_{j \neq i}^{n} \mu_j^i (x_j^1 - x_j^2)) - \|x^1 - x^2\| \\
&= \|x^1 - x^2\| + 2 \cdot \sum_{i=1}^{n} (x_i^1 - x_i^2) \sum_{j \neq i}^{n} \mu_j^i (x_j^1 - x_j^2) - \|x^1 - x^2\| \\
&= 2 \cdot \sum_{i=1}^{n} (x_i^1 - x_i^2) \sum_{j \neq i}^{n} \mu_j^i (x_j^1 - x_j^2)
\end{aligned}$$

In der letzten Zeile sehen wir, dass die Differenz der Abstände von der Korrelation zwischen den Koordinaten abhängt und unterschiedlich stark ausgeprägt ist, je größer die Wertedifferent für die jeweiligen Koordinaten ist. Diesen additiven Term werden wir als Fehlermaß für die vektorielle Repräsentation ansehen.

Da wir gesehen haben, wie stark der Einfluss korrelierter Koordinaten ist, werden wir uns anhand einiger Praxisbeispiele ansehen, wie stark einige Koordinaten tatsächlich korreliert sind.

Reuters Newswire Artikel Die gängige Darstellungsform von Text in einem vektoriellen Kontext ist als Termhäufigkeitsvektoren. In diesem Beispiel betrachten wir den Reuters-21578 Testdatensatz (Lewis 1987). Um zu einem Maß für die Korrelation der einzelnen Terme zu kommen berechnen wir für die ersten 1.000 Dokumente die Termhäufigkeitsvektoren und, ausgehend von der Menge aller dieser Termhäufigkeiten, dann die Korrelation der 1.000 häufigsten Terme. Diese Häufigkeiten entsprechen den oben beschriebenen Messungen μ_j^i. Dabei ist zu beachten, dass die Termhäufigkeiten nicht die Messungen selbst sind, sondern nur eine Komponente einer Messung. Um dies zu verdeutlichen schauen wir uns in Tabelle 4.1 drei intuitive Beispiele an. In all diesen Beispiel sind die Terme stark korreliert, was zu einem großen Fehler führt. Sehr deutlich wird der Einfluß der Korrelation bei den Termen „new“ und „york“. Diese zwei Worte bilden den Namen der Stadt „New York“ und treten damit sehr häufig gemeinsam auf. Falls ein Text nicht den Term „york“ enthält wird er mit großer Wahrscheinlichkeit auch nicht den Term „new“ enthalten. Ein Unterschied in dem Städtename „New York“ würde daher zweimal gewertet.

Term								
	the	said	for	have	with	that	but	some
and	0.7500	0.7150	0.6135	0.5742	0.5712	0.5709	0.5400	0.5210
	york	the	for	and	said	venezuela	over	times
new	0.6418	0.3957	0.3820	0.3791	0.3315	0.3309	0.3203	0.3081
	day	per	ceiling	prices	spot	through	follows	but
price	0.2795	0.2752	0.2605	0.2585	0.2525	0.2519	0.2456	0.2449

Tabelle 4.1: Drei Terme und deren acht am stärksten korrelierte Terme. Diese Terme und deren Korrelation wurde aus den ersten 1.000 Dokumenten des Reuters-21578 Testdatensatzes berechnet. Schön zu sehen ist in dieser Tabelle die Korrelation bekannter Namen und Sequenzen, wie zum Beispiel „New York“, „and the“ oder auch „price per“. Würde man davon ausgehen, dass die Termhäufigkeiten unkorreliert sind, wie es üblicherweise der Fall ist, so führt dies zu fehlerhaften Berechnungen.

Bilddaten Ein weiteres Beispiel für den negativen Einfluß korrelierter Koordinaten auf Vektorraumoperationen wird in der bekannten Arbeit „Eigenfaces for Recognition“ von Turk und Pentland (1991) behandelt. Dort werden die Eigenvektoren der Kovarianzmatrix einer Menge von Bildern als Basis eines neuen Datensatzes mit unkorrelierten Koordinaten verwendet. Das Ergebnis dieser Projektion ist eine drastische Verbesserung der Erkennungsrate. Bei diesem Beispiel müssen wir jedoch vorsichtig sein und nicht die Korrelation der Koordinaten mit der Korrelation der Messungen verwechseln. Durch die Hauptachsentransformation wurde zwar die Korrelation der Kooridanten behoben, die der Messungen blieb jedoch bestehen. Um die Korrenaltion der Messungen zu beheben benötigen wir ein Verfahren, dass diese Information zur

Berechnung einer neuen Repräsentation verwendet (wie es beim später vorgestellten „Tightening“ der Fall ist) und nicht nur eine andere Darstellung bestimmt.

Zeitreihendaten Ebenfalls ein Beispiel für nichtvektorielle Daten die, wenn sie naiv als vektoriel behandelt werden, zu fehlerhaften Vektorraumoperationen führen, sind Zeitreihen. Für diese stellt weniger Korrelation, als mehr die variierende Sematik der Koordinaten ein Problem dar. Es ist allseits bekannt, dass bei den meisten Zeitreihen, wie zum Beispiel Sprache, Biomedizinische- oder Finanzmarktdaten, die Zeitpunkte an denen Ereignisse eintreten oder die Geschwindigkeit mit der diese Auftreten variieren. Diese nichtlinearen Fluktuationen der Zeitachse (Sakoe und Chiba 1978) können zu einer Phasenverschiebung, einer Frequenzverschiebung oder einer Verschiebung beider Aspekte führen. Abbildung 4.3 demonstriert dies für zwei unterschiedliche Datentypen.

Um Fluktuationen der Zeitachse zu behandeln, muss man die Kurven auf ein entsprechendes Ereignis hin normalisieren. Dieser Vorgang, auch Registrierung genannt, gleicht den zeitlichen Verlauf von Abläufen einander an, sodass wichtige Ereignisse immer zum gleichen Zeitpunkt stattfinden. Dafür benötigt man jedoch charakteristische Ereignisse, die als Referenz dienen. Fehlen diese, kann man auch nicht bezüglich dieser normalisieren, was das Angleichen mehrere Kurven fast unmöglich macht. Gerade das Beispiel mit dem dritten Herzschlag verdeutlicht dies, da er nur in Geräuschen kranker Herzen präsent ist, jedoch nicht in denen gesunder. Für das Unterscheiden von kranken und gesunden Herzen ist aber ein normalisierter Zeitverlauf aller Geräusche notwendig. In so einem Fall ist eine sinnvolle Registrierung sehr schwierig bis unmöglich. Abbildung 4.4 zeigt jedoch, wie man einen solchen Fall mit Hilfe eines RDS behandeln kann. Genaueres werden wir in dem nächsten Abschnitt sehen.

Für alle gerade aufgeführten Beispiele gilt, dass wir mit den im Folgenden vorgestellten Methoden die gerade präsentierten Schwierigkeiten umgehen können. Das bald vorgestellte Tightening reduziert die Korrelation zwischen den Koordinaten, und das Registrierungsproblem wird durch einen geeigneten Kernel und die Registrierung bezüglich der Wörterbuchelemente behoben. Mit Hilfe des RDS erhalten wir also korrekt registrierte und unkorrelierte Daten, die problemlos in einem Verktorraumkontext verwendet werden können ohne dass man Gefahr läuft, dadurch Fehler einzuführen.

4.4 RDS Vektorräume

Konkret betrachtet ist ein RDS ein Koordinatenvektorraum, dessen Koordinaten die Koeffizienten von Linearkombinationen aus Wörterbuchelementen $\{d^i\}_{i\in\mathbf{I}} = \mathcal{D}$[3] bil-

[3] In diesem Abschnitt werden wir noch die Wörterbuchelemente als vektoriell ansehen, das heisst wir erwarten dass sie den Vektorraumaxiomen entsprechen. Später werden wir jedoch sehen, dass die nicht notwenig ist und wir auch mit nichtvektoriellen Wörterbuchelementen arbeiten können.

den. Diese Linearkombinationen formen Elemente des Datenraums $\mathcal{X}$, aus dem auch die Wörterbuchelemente stammen[4]:

$$v \in \mathcal{R},\ d^i \in \mathcal{D} \subset \mathcal{X} \quad \text{und} \quad x = \sum_{i \in \mathbf{I}} v_i d^i \in \mathcal{X}.$$

Aufgrund der Redundanz in dem Wörterbuch können komponentenweise verschiedene RDS Elemente den selben Vektor repräsentieren

$$u, v \in \mathcal{R} \quad \text{mit} \quad v_i \neq u_i \text{ für einige } i \in \mathbf{I} \quad \text{und} \quad x = \sum_{i \in \mathbf{I}} v_i d^i = \sum_{i \in \mathbf{I}} u_i d^i.$$

Die RDS-Elemente u und v werden daher als äquivalent angesehen. Ein RDS ist somit der Quotientenraum $\mathcal{R} = \mathbf{R}^n / \sim$ eines Koordinatenvektorraums mit Dimension $n = |\mathcal{D}|$. Die Äquivalenzrelation dieses Quotientenraums ist durch die Äquivalenz innerhalb des Datenraums

$$u \sim v \quad \text{iff} \quad \sum_{i \in \{1 \ldots n\}} v_i d^i = \sum_{i \in \{1 \ldots n\}} u_i d^i$$

gegeben. Aufgrund der Tatsachen, dass ein RDS ein Quotientenraum eines Vektorraums ist, ist er selbst auch ein Vektorraum (Rudin 1991, Kapitel 1, Abschnitt *Quotient spaces*).

Die Idee des RDS ist es, Daten mit Hilfe eines redundanten Wörterbuchs darzustellen. Dies bedeutet aber auch, dass Operationen auf den RDS-Elementen die selben Effekte haben, wie auf den Daten-Elementen selbst. Der RDS soll dabei nur eine andere Sichtweise auf die Daten zeigen. Da wir uns hier hauptsächlich mit Vektorräumen beschäftigen, betrachten wir hier auch nur Vektorraumoperationen wie Addition und skalare Multiplikation sowie die Berechnung eines Skalarprodukts. Wir werden dafür nun ein Skalarprodukt auf Redundante-Wörterbuchräumen (Redundant Dictionary Inner Product - RDIP) betrachten. Es ist so konstruiert, dass Abhängigkeiten zwischen den Wörterbuchelementen mit in die Berechung einbezogen werden:

Definition 4.2 (Redundant Dictionary Inner Product (RDIP))**.** *Seien $u, v \in \mathcal{R}$ Elemente eines RDS so nennen wir*

$$\langle u, v \rangle_{\mathcal{R}} = \sum_{i \in \{1 \ldots n\}} \sum_{j \in \{1 \ldots n\}} \lambda_{ij} u_i v_j \text{ mit } \lambda_{ij} = k(d^i, d^j) \sim \langle d^i, d^j \rangle$$

ein Skalarprodukt über $\mathcal{R}$.

Eine etwas kürzerer Variante das RDIP zu beschreiben, ist mit Hilfe der Gramschen Matrix G des Wörterbuchs:

$$\langle u, v \rangle_{\mathcal{R}} = u^T \cdot G \cdot v \quad \text{mit} \quad G = (\langle d^i, d^j \rangle)_{i,j \in \{1 \ldots n\}}.$$

[4]Der Datenraum kann ein Vektorraum sein, wir werden dies der Einfachheit halber für diesen Abschnitt annehmen, muss es aber nicht.

Der RDS $\mathcal{R}$ – ein Quotientenraum, gegeben durch die Äquivalenzklasse der Koordinatenvektoren die das selbe Datenraumelement ergeben – und das RDIP $\langle \cdot, \cdot \rangle_{\mathcal{R}}$ beschreiben einen Vektorraum mit Skalarprodukt. Ist der Datenraum ein Hilbertraum, so ist auch der RDS ein Hilbertraum.

Wie wir nun im nachfolgenden Kapitel sehen werden, sind wir bei dem Datenraum nicht auf Vektorräume begrenzt. Wir können mit jedem beliebigen Datenraum arbeiten, solange eine Kernfunktion (siehe Abschnitt 4.5.3) auf den Daten definiert ist. Dieses Vorgehen erlaubt es uns, nichtvektorielle Daten im Rahmen eines Vektorraums zu behandeln.

4.5 Die Projektion vektorieller und nicht vektorieller Daten auf Vektorräume, basierend auf redundanten Wörterbüchern

Bisher sind wir stillschweigend davon ausgegangen, dass unsere Daten schon in dem RDS sind und wir sie einfach nur noch analysieren müssen. Wie genau jedoch die Daten in den RDS kommen, werden wir in diesem Abschnitt sehen. Dabei werden wir auch die Behandlung nichtvektorieller Daten näher unter die Lupe nehmen. Gerade was Kompressionsverfahren angeht, so ist doch ein großer Teil dieser Verfahren für nichtvektorielle Daten gedacht. Abgesehen davon erlaubt uns diese Betrachtung die Entwicklung eines Verfahrens zur allgemeinen Behandlung nichtvektorieller Daten innerhalb von Vektorräumen.

Der erste Schritt bei der Arbeit mit redundanten Wörterbüchern ist die Erstellung des Wörterbuchs selbst. Wir werden einfach eine ausreichende Anzahl[5] an Datenelementen zufällig aus der Datenmenge auswählen und diese als Wörterbuchelemente nutzen. Die ausgewählten Elemente sind mit großer Wahrscheinlichkeit nicht linear unabhängig, was die Verwendung der nun beschriebenen Methoden notwendig macht. Einen Überblick über die zur Projektion notwendigen Schritte erhalten wir in Tabelle 4.3, die Details schauen wir uns nun an.

4.5.1 Projektion

Projiziert wird üblicherweise auf Vektorräumen zu denen eine Basis $\mathcal{B} = \{b^1, b^2, ...\}$ gegeben ist. Dazu berechnet man zu dem zu projezierenden Datenelement x das Skalarprodukt mit jedem einzelnen Basiselement b^i. Dies führt zu einem Koordinatenvektor mit genauso vielen Einträgen wir Elementen in der Basis:

$$(v_1, v_2, ...) = (\langle x, b^1 \rangle, \langle x, b^2 \rangle, ...) \tag{4.3}$$

[5] Was genau aussreichend ist, bestimmen wir mit der „Elbow“-Methode (Tenenbaum et al. 2000) die kurz in Abbildung 5.11 beschrieben ist.

Falls die Basis orthonormal ist, könner wir aus den so gewonnenen Koeffizienten direkt das originale Datenelement als gewichtete Linearkombination der Basiselemente $x = \sum v_i b^i$ wiederherstellen.

Im Falle eines redundanten Wörterbuchs verhält es sich ähnlich: wir berechnen das Skalarprodukt zwischen dem Datenelement und jedem einzelnen Wörterbuchelement und erhalten auf diese Weise einen Koordinatenvektor mit genau so vielen Einträgen wie es Wörterbuchelemente gibt. Diese Koordinatenrepräsentation erlaubt jedoch keine einfache Rekonstruktion des ursprünglichen Datenelements mehr. Wir müssen daher einen weiteren Verarbeitungsschritt durchführen, um eine exakte Repräsentation und damit auch eine exakte Rekonstruktionsmöglichkeit zu erhalten. Diesen Schritt nennen wir „Tightening“, da dabei die Repräsentation von Redundanzen befreit und sinnbildlich festgezurrt wird.

4.5.2 Tightening

Aufgrund der Redundanzen in einem Wörterbuch – seine Einträge sind meistens weder orthogonal noch linear unabhängig – führt eine wie oben beschriebene Projektion zu Koordinatenvektoren, die voller Abhängigkeiten zwischen den einzelnen Koordinaten sind. Ein kurzes Beispiel soll dies nun verdeutlichen.

Beispiel 4.1. *Betrachten wir ein Wörterbuch $\mathcal{D} = \mathcal{B} \cup \{\hat{b^n}\} = \{b^1, ...b^n, \hat{b^n}\}$ mit $n+1$ Einträgen wobei b^n und $\hat{b^n}$ komponentenweise identisch sind*[6]*. Bei der Projektion eines Datenelements x auf dieses Wörterbuch erhalten wir einen Koordinatenvektor*

$$(\langle x, b^1\rangle, \langle x, b^2\rangle, ..., \langle x, b^n\rangle, \langle x, \hat{b^n}\rangle) = (v_1, v_2, ...v_n, \hat{v_n}) = (v_1, v_2, ...v_n, v_n)$$

mit zwei identischen Einträgen v_n und $\hat{v_n}$. Bei einer Rückprojektion

$$x' = \sum_{i=1}^{n} v_n b^n + \hat{v_n}\hat{b^n} = x + v_n b^n$$

wird nun der Anteil v_n des Wörterbuchelements b^n an dem Datenvektor x doppelt gewichtet, was zu einer falschen Rekonstruktion führt.

Von der fehlerhaften Repräsentation ist jedoch nicht nur die Rekonstruktion betroffen. Auch Addition oder skalare Multiplikation führen zu fehlerhaften Ergebnissen. Bei der Berechnung des Skalarprodukts führt eine fehlerhafte Repräsentation dazu, dass der Einfluss redundanter Wörterbuchelemente überproportional hoch gewichtet wird. Aus diesem Grund müssen wird die Abhängigkeiten zwischen den einzelnen Wörterbucheinträgen aus der Repräsentation herausrechnen. Eine Möglichkeit hierfür bietet Matching Pursuit. Wie schon besprochen, berechnet dieses Verfahren eine dünne Repräsentation eines Koordinatenvektors. Als Ausgangsvektor können wir hier den Koordinatenvektor ansetzen, den wir durch Projektion auf das Wörterbuch erhalten

[6] Zugunsten eines besseren Verständnisses erlauben wir uns hier einen etwas laxeren Umgang mit dem Konzept der Mengen.

haben. Eine andere Sichtweise ist, dass Matching Pursuit das Gram-Schmidtsche Orthogonalisierungsverfahren auf den Koordinaten anstatt auf dem Wörterbuch anwendet werden. Es liefert uns also eine Koordinatenrepräsentation, die der einer orthogonalen Basis entspricht. In diesem Kontext bietet Tabelle 4.2 nochmals eine Überblick über Matching Pusuit.

Schritt	Matching Pursuit	
1	Initialize	Initialisiere den Ausgabevektor $v' = (0, 0, ...)$ mit Nullen.
2	Best Match	Finde i sodass $i = \text{argmax}_{\in\{1...m\}}\|v_j\|$.
3	Update	Setze $v'_i = v_i$, $v_i = 0$ und für alle $j \neq i$ setze $v_j = v_j - v_i \cdot G_{ij}$.
4	Iterate	Wiederhole 3 und 4 solange die Anzahl der nicht-null Einträge in v' kleiner als k und $\|v_i\| \geq \epsilon$ ist.

Tabelle 4.2: Die Addaption des Matching Pursuit Verfahrens auf den nichtvektoriellen Fall. Als Eingabe für den Algorithmus dient die RDS Projektion $v = (k(x, d_1), k(x, d_2), k(x, d_3), \ldots, k(x, d^m))$ des Datenelements x, die Gramsche Matrix $G = (\lambda_{ij}) := \nu_i \nu_j k(d_i, d_j)$ des normalisierten Wörterbuchs $\mathcal{D}$ sowie die Anzahl der erwarteten Nicht-Null-Einträge k und den Fehler ϵ, den wir beim Tightening tollerieren.

4.5.3 Skalarproduktkernel

Die Projektion eines Datenelements auf ein RDS erfordert die Berechnung eines Skalarprodukts zwischen dem Datenelement und jedem Wörterbuchelements. Für nichtvektorielle Daten stellt dies jedoch ein Problem dar, da auf diesen Daten eigentlich kein Skalarprodukt definiert ist. Es ist jedoch in sehr vielen Fällen ein sogenannter Skalarproduktkernel (engl. Inner Product Kernel) definiert.

Der Begriff Kernel[7] stammt ursprünglich aus der Theorie der Integralgleichungen. Dort wurden Kernel verwendet, um im Rahmen von Integraloperatoren Funktionen aus einer Domäne in eine andere zu überführen. In der Mustererkennung wird eine spezielle Eigenschaft dieser Kernel in den Vordergrund gesetzt: die Berechnung eines Skalarprodukts in virtuellen, hochdimensionalen Räumen. Diese, auch als Mercer-Kernel bekannte, Funktionen werden daher auch Skalarproduktkernel genannt.

Definition 4.3 (Kernfunktion)**.** *Eine symmetrische und quadratisch integrierbare Funktion $K(x, y)$ mit $x, y \in \mathbf{R}^n$ und $n \in \mathbf{N}$ heißt Kernel oder Kernfunktion.*

Solch eine Kernfunktion kann nach Schmidt (1907) auch als eine Art Skalarprodukt in einem Hilbertraum angesehen werden. Dazu werden die Funktionswerte x und

[7] Im Deutschen wäre eigentlich der ursprünglich deutsche Begriff Kern üblicher. Der Weg des Kerns führte jedoch von den Integralgleichungen über die Statistische Lerntheorie hin zu den maschinellen Lernverfahren und damit ins Englische. Wir werden daher meistens den englischen Begriff „Kernel“ verwenden, manchmal jedoch auch Kern oder Kernfunktion.

y, mit Hilfe der Projektion $\varphi_1(x), ...\varphi_n(x)$ in einen Hilbertraum projiziert in dem dann das Skalarprodukt (mit Gewichtungsfaktoren μ_i)

$$\sum_{i \in \mathbf{I}} \mu_i \varphi_i(x) \varphi_i(y)$$

berechnet wird. Wir werden auf einen Beweis des als Hilbert-Schmidt Theorem bekannten Satzes verzichten.

Satz 4.3 (HilbertSchmidt Theorem – Riesz und Sz.-Nagy (1990); Schmidt (1907))**.** *Jede Funktion $K(x, y)$ die symmetrisch und quadratisch integrierbar ist, kann als Reihe der Form*

$$K(x, y) = \sum_{i \in \mathbf{I}} \mu_i \varphi_i(x) \varphi_i(y) \tag{4.4}$$

entwickelt werden. Die Reihe ist im Mittel konvergent. Die $\{\varphi_i\}$ beschreiben die orthonormale Folge charakteristischer Funktionen mit $\{\mu_i\}$ als charakteristischen Werten der Transformation K beschrieben durch den Kern $K(x, y)$.

Ein weiterer wichtiger Satz in diesem Kontext ist das Mercer Theorem. Es garantiert uniforme Konvergenz und damit eine Existenz des erwähnten Skalarprodukts. Auch bei diesem Satz werden wir auf einen Beweis verzichten.

Satz 4.4 (Mercer Theorem – Riesz und Sz.-Nagy (1990))**.** *Falls die Transformation K, erzeugt von einem kontinuierlichen symmetrischen Kern, positiv ist, d.h. $\langle Kf, f\rangle \geq 0$ für alle f oder alle $\mu_i \neq 0$ sind positiv, dann ist die Entwicklung (4.4) uniform konvergent.*

Im Gegensatz zum üblichen Skalarprodukt sind Kernel nicht zwangsläufig auf vektorielle Daten beschränkt. Es gibt zahlreiche Kernel für strukturierte, semi strukturierte und unstrukturierte Daten (Gärtner 2003), Beispiele hierfür sind string kernels (Lodhi et al. 2002) für Text, graph kernels (Vishwanathan et al. 2010) für Graphen oder baumähnliche Strukturen und Local Alignment Kernels für biologische Sequenzen (Vert et al. 2004). Diese Kernel erlauben es, eine Art Skalarprodukt von zwei nichtvektoriellen Daten zu berechnen. Dabei werden die Daten implizit von einer abstrakten Transformation Φ in einen Vektorraum überführt und dort dann das Skalarprodukt berechnet.

$$(v_1, v_2, ...) = (\langle \Phi(x), \Phi(b^1)\rangle, \langle \Phi(x), \Phi(b^2)\rangle, ...) = (k(x, b^1), k(x, b^2), ...)$$

Auf diese Weise können wir nun auch das Skalarprodukt von nichtvektoriellen Daten berechnen und die Projektion von nichtvektoriellen Daten in ein RDS durchführen.

4.5.4 Mathematische Demonstration

In diesem Abschnitt wollen wir uns die genaue Funktionsweise der gerade beschriebenen Methode ansehen und zwar am Beispiel des Raums der Polynome. Der besseren Übersichlichkeit wegen beschränken wir uns auf Polynome bis zum ersten Grad: $\mathcal{D} = \{d_1, d_2, d_3\}$, $d_1(x) = \sqrt{1/2}$, $d_2(x) = \sqrt{3/2}x$, $d_3(x) = \sqrt{3/8}(x - 1)$. Diese

Schr.	RDS Projektion	
1	Auswahl des Wörterbuchs	Wähle zufällig m Datenelement aus, erstelle damit das Wörterbuch $\mathcal{D} = \{d^1, d^2, \dots d^m\}$ und berechne für jedes Wörterbuchelement eine Normalisierungskonstante $\nu_i = 1/\sqrt{k(d^i, d^i)}$.
2	Projektion auf den RDS	Berechne für jedes Datenelement x^i dessen Projektion auf den RDS $x \rightarrow v = (\nu_1 \cdot, k(x, d^1), \nu_1 \cdot, k(x, d^2), \nu_1 \cdot, \dots, k(x, d^m))$.
3	Tightening im RDS	Zurre die redundante Repräsentation v fest, indem davon die Matching Pursuit Repräsentation v' im RDS berechnet wird.
4	Vektorraumberechnungen	Nutze die RDS Repräsentationen v' anstelle der Datenraumelemente x um Vektorraumberechnungen durchzuführen.

Tabelle 4.3: Die RDS Projektion und das Festzurren des Datensatzens $\mathcal{X} = \{x^1, \dots x^n\}$ um ihn mit Vektorraumoperationen nutzen zu können.

spannend einen Vektorraum der Dimension 2 auf. Wir werden dabei nur Funktionen in diesem Raum betrachten, die auf dem Intervall $[-1, 1]$ definiert sind. Schauen wir uns also folgende Funktionen und deren Repräsentationen im RDS an: $f_1(x) = x + 2$, $f_2(x) = 2x - 2$ und $f_3 = x$. Das Skalarprodukt in diesem Raum ist definiert als

$$\langle f, g \rangle = \int_{-1}^{1} f(x) \cdot g(x) dx$$

und die Gramsche Matrix ist somit

$$G = (\langle d_i, d_j \rangle)_{i,j=1\dots3} = \begin{pmatrix} 1 & 0 & -\sqrt{\frac{3}{4}} \\ 0 & 1 & \frac{1}{2} \\ -\sqrt{\frac{3}{4}} & \frac{1}{2} & 1 \end{pmatrix}.$$

Eine Projektion der Funktion f_1 auf den Koordinatenvektorraum des Wörterbuchs ergibt folgenden Koordinatenvektor

$$v_1 = (2\sqrt{2}, \frac{1}{2}\sqrt{6}, -\frac{5}{6}\sqrt{6}).$$

Wie die Gramsche Matrix zeigt, besitzt dieser Vektor Abhängigkeiten zwischen der ersten und der dritten Koordinate. Daher benötigen wir das Matching Pursuit Verfahren, um diese Abhängigkeiten zu entfernen und eine „tighte“ Repräsentation zu erhalten:

$$v_1' = (2\sqrt{2}, \frac{1}{2}\sqrt{6}, 0).$$

Diese entspricht exakt der (Polynomraum-) Datenraumdarstellung der Funktion:

$$f_1' = 2\sqrt{2} \cdot \sqrt{1/2} + \frac{1}{2}\sqrt{6} \cdot \sqrt{3/2}x + 0 \cdot \sqrt{3/8}(x-1) = 2 + x.$$

Auf die gleiche Weise verhält es sich auch mit f_2 wobei die Projektion und die Matching Pursuit Repräsentation wie folgt aussieht:

$$v_2 = (2\sqrt{2}, \frac{1}{3}\sqrt{6}, \frac{4}{3}\sqrt{6}) \text{ und } v_2' = (0, 0, \frac{4}{3}\sqrt{6}).$$

Wenn wir nun die „tighten“ Repräsentationen der beiden Funktionen addieren, so erhalten wir

$$v_a' = v_1' + v_2' = (2\sqrt{2}, \frac{1}{2}\sqrt{6}, \frac{4}{3}\sqrt{6})$$

was $f_a(x) = 3x$ entspricht. Auf der anderen Seite, wenn wir die Repräsentation von $f_3(x) = x$ berechen, so erhalten wir

$$v_3' = (0, \frac{1}{3}\sqrt{6}, 0).$$

Dies ist der Tatsache geschuldet, dass das Wörterbuch redundant ist und mehrere gleichberechtigte, Repräsentationen einer Funktion zulässt. Dies wird besonders deutlich wenn man sich die Repräsentation zu dem Vektor $\mathbf{0}$ ansieht. Ganz offensichtlich kann dies jedes Vielfache des Vektors

$$v_0' = (2\sqrt{2}, -\frac{2}{3}\sqrt{6}, \frac{4}{3}\sqrt{6})$$

sein. Da es sich bei dem RDS um einen Quotientenraum handelt, ändert sich die korrespondierende Funktion eines Vektors nicht, wenn wir v_0' oder ein beliebiges Vielfaches davon hinzuaddieren. Wenn wir das Skalarprodukt bilden, sehen wir sofort, dass $v_0' \cdot G = 0$ ergibt und somit für einen beliebigen Vektor u, $(u + v_0')^T \cdot G = u^T \cdot G$. Für zwei Vektoren u und v und zwei Skalare α und β erhalten wir

$$\langle u + \alpha v_0', v + \beta v_0' \rangle = (u + \alpha v_0')^T G (v + \beta v_0') = u^T G v.$$

4.6 Kompressionsbasierte Mustererkennung

In den vorangegangenen Kapiteln haben wir uns eine ganze Menge Wissen über Kompression und Kompressionsverfahren erarbeitet. In diesem Kapitel beschäftigten wir uns bis jetzt mit allgemeinen Herangehensweisen zur Bestimmung dünner oder komprimierter Repräsentationen. Im Folgenden werden wir den Kompressionsaspekt stärker in unser Blickfeld nehmen.

Wie bereits im Kapitel zur NCD erwähnt, basieren fast alle aktuell verfolgten Ansätze zur kompressionsbasierten Mustererkennung auf der Annahme eines durch die

Kolmogorov Komplexität gegebenen absoluten Maßes, das jedes andere Maß minorisiert (Cilibrasi und Vitànyi 2005). Dieses Maß definiert auch einen Raum, basierend auf Ähnlichkeitsmerkmalen, in dem Ähnlichkeiten zwischen Daten absolut berechnet werden können. Cilibrasi und Vitànyi (2005, 3.2 Quasi-Universality) geben jedoch auch an, dass diese Aussagen nur in der Theorie (für unendlich lange Zeichenketten) haltbar sind. Für endliche Zeichenketten würden nur *einfache* Merkmale zur Berechnung der Ähnlichkeiten herangezogen. Genau an diesem Punkt werden wir nun einhaken. Wir werden im Folgenden eine Theorie entwickeln, die es erlaubt, basierend auf bestehenden Überlegungen zu Kompressionsverfahren, die Ergebnisse der NCD zu erklären. Dabei soll jedoch nicht die Kolmogorov-Komplexität als Begründung herangezogen werden, sondern ein Aspekt, der fast allen Kompressionsverfahren innewohnt. Wir werden unsere Theorie auf der Überlegung aufbauen, dass alle betrachteten Kompressionsverfahren mit Hilfe redundanter Wörterbücher Daten in eine dünne Repräsentation überführen.

Wie sollte nun so eine Theorie aussehen? Basierend auf Rudins Einführung der Distributionen (Rudin 1991) werden wir erst eine Liste mit Anforderungen aufstellen, welche von der Theorie erfüllt werden sollen:

1. Alle zur Mustererkennung verwendeten Kompressionsalgorithmen sollten sich mit Hilfe der Theorie erklären lassen.

2. Alle experimentellen Ergebnisse sollten sich mit Hilfe der Theorie reproduzieren lassen.

3. Die Theorie sollte es ermöglichen, alle Ergebnisse (egal ob Erfolg oder Misserfolg) fundiert zu erklären.

4. Die Theorie sollte an bestehende Erkenntnisse aus dem Bereich der Mustererkennung anknüpfen.

Die erste Anforderung soll eine Verknüpfung zwischen der Theorie und den bestehenden Kompressionsverfahren herstellen. Sie fordert, dass existierende Verfahren beschrieben werden können und somit überhaupt die Möglichkeit besteht, Ergebnisse zu erklären. In den folgenden Abschnitten werden wir Definitionen für Kompressoren entwickeln und diese durch prägnante Beispiele mit real existierenden Verfahren verknüpfen.

Der zweite Punkt fordert, dass existierende Ergebnisse sich durch die Theorie reproduzieren lassen. Hier könnte man argumentieren, dass bereits die erste Anforderung dies erfüllt, da sich ja die Kompressoren durch die Theorie erklären lassen und somit die Ergebnisse ebenso. Wir werden jedoch einen Schritt weiter gehen und schauen, ob sich nicht auch mit Hilfe von sehr einfachen und intuitiven (der Theorie entsprechenden) Verfahren die gleichen Ergebnisse erzielen lassen. In Kapitel 5 werden wir anhand einiger prägnanter Beispiele diese Fähigkeiten prüfen.

Als Basis für eine solche Struktur benötigen wir jedoch zuerst eine genaue Vorstellung davon, welche Eigenschaften Kompressoren, und natürlich auch die dazugehörigen Dokumente überhaupt haben. Wir werden uns daher in diesem Abschnitt erst einmal den zugrunde liegenden Definitionen widmen. Aufbauend auf diesen werden wir dann einige (theoretische) Ergebnisse untersuchen, die wir direkt aus den Definitionen gewinnen können. Im nachfolgenden Kapitel werden wir uns dann intensiver mit den praktischen Aspekten der hier entwickelten Verfahren beschäftigen. Die hier vorgestellten Ideen sind denen von Sculley und Brodley (2006) sehr ähnlich. Wir werden jedoch deutlich tiefer eindringen in die Analyse kompressionsbasierter Mustererkennung.

Beginnen wir mit der Definition eines Dokuments. Hier geht es darum, klarzustellen, worauf genau die Kompressionsverfahren arbeiten werden.

Definition 4.4 (Dokument)**.** *Sei $\mathbf{D} \subset \mathcal{H}$ eine Teilmenge eines Hilbertraums reelwertiger Funktionen mit kompaktem, reellem $\mathbf{K} \subset \mathbf{R}$ oder diskretem $\mathbf{K} \subset \mathbf{I}$ Definitionsbereich, so nennen wir jedes Element $d \in \mathbf{D}$ eines solchen Raums Dokument.*

Basierend auf dieser Definition können wir auch gleich den dazugehörigen Kompressor definieren.

Definition 4.5 (Kompressor)**.**
Sei $C : \mathbf{D} \mapsto \mathcal{G} : f \mapsto x_f$ ein Operator, der den Elementen eines Hilbertraums $\mathbf{D} \subseteq \operatorname{span} \mathcal{D}$ ihre entsprechenden Sequenzraumrepräsentation $x \in \mathcal{G}$ zuweist. Sei $D : \mathcal{G} \mapsto \mathbf{D} : x \mapsto f_x$ ein Operator, der die (nicht notwendigerweise exakte) Rekonstruktion durchführt. Falls die Sequenzraumrepräsentation, für $0 \leq \epsilon \in \mathbf{R}$ und $r_f \in \mathcal{H}$ mit $||r_f||_2 = 1$, von dem zugrundeliegenden Wörterbuch $\mathcal{D} = \{\phi_i\}_{i \in \mathbf{I}}$ in folgender Weise abhängt:

$$C(f) = x_f \qquad \textit{sodass} \qquad f = D(x_f) + \epsilon \cdot r_f = \sum_{i=1}^{n} \phi_i \cdot x_f^i + \epsilon \cdot r_f$$

und

$$||C(f)||_0 \leq ||f||_0, \quad \epsilon \to 0 \quad \textit{als} \quad n \to |\mathbf{I}|,$$

ist, dann ist C ein Kompressor.

In dieser Definition ist, im Fall einer verlustbehafteten Kompression, ϵ der Kompressionsfehler und r_f der normierte Residuumsvektor. Die Größe der Kompression $||C(f)||_0$ hängt dabei maßgeblich von dem Kompressor C, der Funktion f und der Exaktheit der Repräsentation – oder anders formuliert dem Rekonstruktionsfehler ϵ – ab. Der Grad der Kompression wird durch die Nicht-Nullstellen der Repräsentation von f mit Hilfe von $\mathcal{D}$ bestimmt. In diesem Zusammenhang sollte auch noch erwähnt werden, dass jeder Kompressor per Definition jede Funktion beliebig genau darstellen kann. Dies liegt in der Tatsache begründet, dass der Dokumentenraum der Aufspann des Wörterbuchs ist. Bei der verlustbehafteten Kompression wird diese Eigenschaft zugunsten einer besseren Kompression oft vernachlässigt.

Die Definition trifft keine Aussage darüber wie das Wörterbuch angegeben ist, etwa implizit als eine Funktionsklasse oder explizit als eine klar definierte Menge von

Funktionen. Praktisch bedeutet dies, dass es sowohl möglich ist, das Wörterbuch explizit abzuspeichern, als auch als Funktionen immer aktuell zu berechnen. Dieser Unterschied ist gerade bei Vergleichen diskreter und kontinuierlicher Funktionen zu beachten. Im ersten Fall wird das Wörterbuch mit abgespeichert, wohingegen im zweiten Fall das Wörterbuch als Funktionenklasse (Sinus- und Cosinusschwingungen zum Beispiel) gegeben ist.

Der Algorithmus von Lempel und Ziv basiert auf der Idee, dass lange und oft wiederkehrende Zeichenketten nur einmal im Wörterbuch gespeichert und von da an nur noch referenziert werden müssen. Auf diese Weise erhält man auch ein Maß für die Komplexität der zu komprimierenden Zeichenkette. Gibt es viele unterschiedliche Referenzen, so kann man von einer hohen Komplexität ausgehen, da der Algorithmus nicht in der Lage war lange, wiederkehrende und zusammenhängende Sequenzen zu finden. Gibt es hingegen nur wenige Referenzen, so war die Zeichenfolge eher homogen. Dies entspricht der von uns gegebenen Definition eines Kompressors insofern, als dass eine gute Kompression das Wörterbuch sehr gut an das Dokument anpassen kann, wohingegen bei einer schlechten Kompression die Funktion nur mit viel Aufwand an das Wörterbuch angepasst werden kann.

Beispiele für Kompressoren

Um ein besseres Verständnis für die oben gegebenen Definitionen zu bekommen, werden wir uns an realen Beispielen anschauen, wie Dokumente und Kompressoren in der Praxis aussehen. Wir werden mit einem intuitiven Beispiel für die verlustbehaftete Kompression im Kontinuierlichen anfangen und dann zwei weitere Beispiele für die diskrete, verlustfreie Kompression betrachten.

Beispiel 4.2 (JPEG)**.** *Im Fall kontinuierlicher, funktionaler Daten ist eine Transformation von einem Funktionenraum in einen anderen eine eher natürliche und seit langem gebräuchliche Sichtweise. In diesem Beispiel sind Funktionen $f \in \mathbf{D} \subset L^2(\mathbf{R})$ quadratisch integrierbar über $\mathbf{R}$. Somit ist es naheliegend, für den Kompressor die Fourier- oder die Cosinus-Transformation anzunehmen. Die Kompression und der Rekonstruierungsfehler entstehen nun dadurch, dass man Frequenzen (oder Basiselemente), die nur einen geringen Beitrag zum Signal leisten, auf null setzt. Der Erfolg dieser Kompressionsmethode basiert maßgeblich auf der Annahme, natürliche Daten seien eher glatt und daher eher niederfrequent, was den oben beschriebenen Transformationen entgegen kommt.*

Beispiel 4.3 (LZ77)**.** *Bei diskreten Daten wie zum Beispiel Text, XML, HTML oder eher exotischen Formaten, wie der DNA, greift man eher zu verlustfreien Verfahren wie der Lempel-Ziv Kompressorfamilie. Allen Kompressionsverfahren dieser Gruppe ist eins gemein: Sie stellen Zeichenketten als Abfolgen von Wörterbuchreferenzen dar. Beim LZ77 ist das Wörterbuch implizit durch den Puffer gegeben, bei Verfahren wie LZ78 oder LZW wird es explizit angegeben.*

Ähnlich wie bei den Funktionenraumelementen betrachten wir die Dokumente als Elemente einens Sequenzraums $\mathcal{H}$, wobei jede Komponente der Sequenz durch ein Zei-

chen repräsentiert wird. Der Kompressor nimmt nun ein solches Dokument und transformiert es in eine andere Repräsentation in $\mathcal{G}$, in der jede Komponente eine Referenz auf ein Wörterbuchelement aus dem Wörterbuch $\mathcal{D}$ aller möglicher Zeichenketten darstellt. Um die direkte Anwendbarkeit der Definition auf den LZ77 Algorithmus zu erhalten, müssen die Positionsinformationen in den Indizes codiert sein. Dies ist vergleichbar mit der Indizierung der Wavelet Koefizienten (Daubechies 2004), bei denen Skalierung und Diletierung Teil der Indizes ist.

Formal bedeutet dies, dass ein Dokument $d = (a, b, c, d, ..., z, ...)$, welches mit Hilfe des Kompressors C_{LZ77} transformiert wurde, folgende Form hat:

$$C_{LZ77}(d) = (x_1, x_2, ..., x_n, ...) \text{ wobei } x_i = 0010001...$$

Dabei bedeutet der Binäreintrag, dass das $i$te Wörterbuchelement ϕ_i an der dritten und der siebten Position in der Zeichenkette auftritt. Falls ein Wörterbuchelement gar nicht in der Zeichenfolge auftritt, wird die Komponente des Vektors an der ensprechenden Stelle eine Null zugewiesen.

Beispiel 4.4 (Lauflängencodierung)**.** *Im Gegensatz zu den LZ-Algorithmen codiert die Lauflängencodierung (englisch Run Length Encoding – RLE) die Referenzen auf das Wörterbuch nicht anhand ihrer Position, sondern anhand ihres Wertes. In dem gerade aufgeführten Beispiel $C_{RLE}(d) = x = (x_1, x_2, ..., x_n, ...)$ würde der resultierende Vektor x von links mit nicht-Null Einträgen der Form $x_i = j$ aufgefüllt. Dabei würde der Index i angeben, dass wir uns die i-te Referenz ansehen und dort das Wörterbuchelement ϕ_j einfügen.*

Was genau diese unterschiedlichen Repräsentationen für die Mustererkennung bedeuten, werden wir in den nun folgenden Abschnitten diskutieren. Zuerst benötigen wir jedoch eine kurze Proposition, die uns die Arbeit mit realen Kompressoren erleichtern wird. Reale Kompressoren setzen sich meist aus einer Reihe unterschiedlicher Kompressoren zusammen und erst deren Kombination bewirkt die letztendliche Kompression. Damit wir auch bei der Kombination von Kompressoren von einem Kompressor sprechen können, betrachten wir nun die folgende Proposition:

Proposition 4.1 (Kombinierbarkeit)**.** *Seien C und C' zwei Kompressoren, dann ist ihre Kombination $C^* = C \circ C'$ ebenfalls ein Kompressor.*

Beweis. Da es sich, wie schon erwähnt bei jedem Sequenzraum ebenfalls um einen Hilbertraum handet, gilt für die Kompression $C^* = C \circ C'$ folgendes: Für jedes $f \in \mathcal{H}$

$$C'(f) = f' \quad \text{mit} \qquad ||C'(f)||_0 \leq \|f\|_0$$

und

$$C(f') = f^* \quad \text{mit} \qquad ||C(f')||_0 \leq \|f^*||_0$$

daher

$$||f||_0 \geq ||C'(f)||_0 \geq ||C(f')||_0 = ||f^*||_0.$$

Der Zusammenhang mit dem Wörterbuch ergibt sich durch

$$\begin{aligned} f =& \mathcal{D}(x_f{}') + \epsilon' \cdot r'_f = \mathcal{D}(\mathcal{D}'(x_f) + \epsilon \cdot r_f) + \epsilon' \cdot r'_f \\ =& \mathcal{D} \circ \mathcal{D}'(x_f) + \epsilon \cdot \mathcal{D}(r_f) + \epsilon' \cdot r'_f \\ =& \mathcal{D}^*(x_f) + \epsilon \cdot \epsilon' \cdot (\mathcal{D}(r_f) + r'_f) = \mathcal{D}^*(x_f) + \epsilon^* \cdot r^*_f \end{aligned}$$

□

Da wir nun alle relevanten Aspekte beisammen haben, werden wir uns nochmals die normalisierte Kompressionsdistanz – mit der urspüngliche Definition 2.13 haben wir uns schon in Kapitel 2 beschäftigt – ansehen. Hier werden wir eine Definition angeben, die zu unserer Definition eines Kompressors passt.

Definition 4.6 (Normalisierte Kompressionsdistanz (NCD))**.**
Sei C ein Kompressor und $d_{NCD} : \mathcal{H} \times \mathcal{H} \mapsto \mathbf{R}$ definiert wie folgt

$$d_{NCD}(f,g) = \frac{||C(f+g)||_0 - \min\{||C(f)||_0, ||C(g)||_0\}}{\max\{||C(f)||_0, ||C(g)||_0\}},$$

dann nennen wir d_{NCD} die normalisierte Kompressionsdistanz.

Wenn wir uns die Definition eines Kompressors $C(f) = c_f$ genau ansehen und $||c_f||_0 < ||c_g||_0$ annehmen, können wir die Definition der NCD umschreiben zu

$$\begin{aligned} d_{NCD}(f,g) &= \frac{||c_{(f+g)}||_0 - \min\{||c_f||_0, ||c_g||_0\}}{\max\{||c_f||_0, ||c_g||_0\}} \\ &= \frac{1}{S}\left(\sum_{i \in \mathbf{I}} |c^i_{(f+g)}|_0 - \sum_{i \in \mathbf{I}} |c^i_f|_0\right) \end{aligned}$$

wobei $S = ||c_g||_0$ ein Normalisierungsfaktor ist. Diese Form legt nahe, dass die NCD einfach nur die Anzahl der Nicht-Null Einträge zählt (aufgrund der ℓ^0 Norm). Sie zieht von der Summe $c_{(f+g)}$ die Anzahl der Einträge der kleineren Funktion ab und gewichtet die Differenz mit der Anzahl der größeren Datei. Dies enspricht, in einer sehr naiven Sichtweise, dem Vergleich der Koeffizienten der zwei Funktionen c_f und c_g.

Da die Kompression der Summe $c_{(f+g)}$ einer dünn besetzten Repräsentation von $f + g$ entspricht, kann man davon ausgehen, dass je ähnlicher sich die beiden Funktionen sind, umso mehr gemeinsame Koeffizienten haben sie. Dies führt dazu, dass nur wenige zusätzliche Koeffizienten benötigt werden, um die Summe zu codieren, als für die größere der beiden Funktionen benötigt werden.

Cilibrasi und Vitànyi (2005, Section 3.2) haben bereits angedeutet, dass die NCD auch als ein Maß für die f und g gemeinsame Information gesehen werden kann. Dies entspricht dem gerade dargelegten Blickwinkel, da hier auch die Information aus f, in unserem Fall die Anzahl der Nicht-Null Einträge, mit der der konkatenierten Funktionen verglichen wird.

Betrachten wir das innere Produkt bezüglich orthogonaler Basen $\{\varphi_i\}_{i\in\mathbf{I}}$ der Koordinaten Vektoren x_f und x_g zweier Funktionen f und g, so erhalten wir das bekannte Konstrukt:

$$\begin{aligned}\langle f, g\rangle &= \sum_{i\in\mathbf{I}} f_i \cdot g_i \\ &= \sum_{i\in\mathbf{I}} \langle f, \varphi_i\rangle \cdot \langle g, \varphi_i\rangle = \langle x_f, x_g\rangle.\end{aligned}$$

Es ist jedoch zu beachten, dass die Koordinaten bezüglich einer nicht redundanten Repräsentation gegeben sind (Riesz und Sz.-Nagy 1990, Section 11 and 33). Die dritte Gleichheit ist durch die Parsevalsche Formel gegeben. Würde es sich bei dem zugrunde liegenden System nicht um eine orthogonale Basis, sondern um ein redundantes Wörterbuch handeln, so wäre die Gleichheit nicht mehr gegeben. Die Komponenten der einzelnen Vektoren wären nicht mehr unabhängig voneinander und Komponenten innerhalb eines Vektors, mit unterschiedlichen Indizes, könnten immer noch einen gemeinsamen Informationsgehalt haben.

$$\langle \phi_i, \phi_j\rangle \neq 0 \quad \text{für} \quad i \neq j.$$

Dies verhindert den Vergleich von redundanten Repräsentationen. Die NCD versucht jedoch genau dies. Die LZ-Verfahren generieren Repräsentationen mit Hilfe hochgradig redundanter Wörterbücher, und die NCD versucht diese zu vergleichen. Daher behalfen sich Cilibrasi und Vitànyi mit der Betrachtung eines abgeleiteten Wertes – der Differenz der Anzahlen der Nicht-Null Einträge. Es wurde also nicht die Ähnlichkeit der Komponenten betrachtet, sondern die Ähnlichkeit der Anzahl der Komponenten.

Wie wir bereits am Anfang dieses Kapitels gesehen haben, gibt es geschicktere Möglichkeiten, redundante Repräsentationen mit Hilfe eines inneren Produkts zu vergleichen. Definition 4.2 gibt uns ein inneres Produkt über redundanten Wörterbüchern. Dieses werden wir im Weiteren verwenden, um kompressionsbasierte Repräsentationen zu vergleichen. Wie wir in dem vorangegangenen Abschnitt gesehen haben, bildet der Raum aller redundanten Repräsentationen einen Hilbert-Raum, wir können also nach Belieben diese Repräsentationen addieren, mit Skalaren multiplizieren und eben auch mit Hilfe des inneren Produkts vergleichen.

4.7 Zusammenfassung

In diesem Kapitel haben wir gesehen, wie wir mit Vektorräumen über redundanten Wörterbüchern arbeiten können. Dabei haben wir, mit Hilfe der Theorie der kompressionsbasierten Mustererkennung, vektorielle und nicht vektorielle Daten innerhalb von vektorraumbasiereten Methoden einsetzen können. Diese Möglichkeiten haben uns direkt zu den komprimierten Repräsentationen geführt. Wie wir erfahren haben, stellten

diese, gleich ob diskret oder kontinuierlich, eine spezielle Form der Vektorräume über redundaten Wörterbüchern dar. Dies erlaubt uns nun jedoch, die für diese Vektorräume gewonnenen Erkenntnisse direkt in der Praxis anzuwenden. Im nun folgenden Kapitel werden wir die Praxistauglichkeit der hier entwickelten Theorie sehen und Einblicke in die Anwendbarkeit der Methode erlangen.

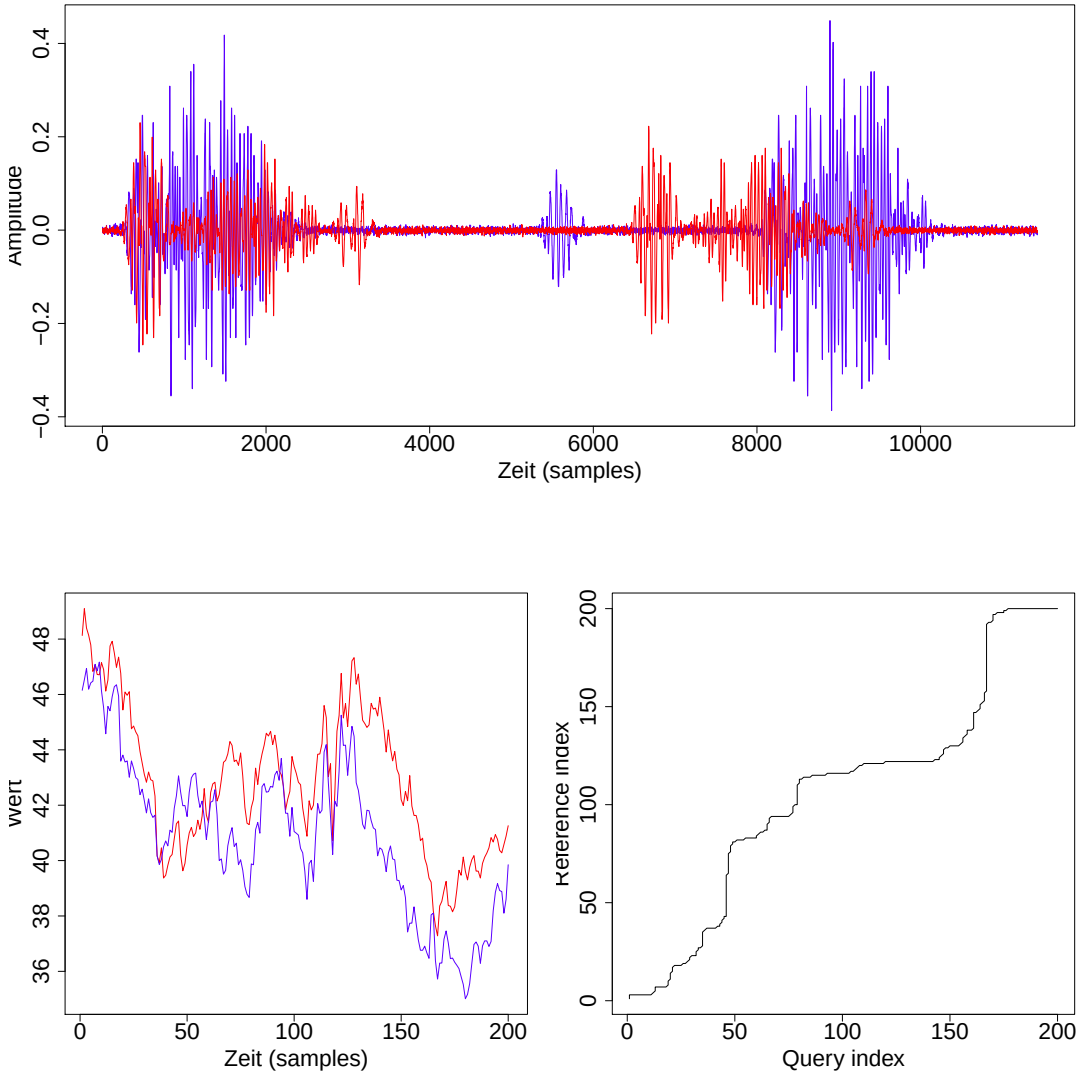


Abbildung 4.3: Zwei Beispiele einer Phasen- und einer Frequenzverschiebung eines Ereignisses einer Zeitreihe. Das Bild auf oben zeigt zwei Herzgeräusche, wobei, aufgrund unterschiedlicher Herzfrequenzen die Zeitpunkte des zweiten Herzschlags variieren. Eine Besonderheit der blauen Kurve ist ein dritter Herzschlag – ein Zeichen für einen existierende Herzfehler – der eine Registrierung der Kurven auf die zwei üblichen Herzschläge sehr schwierig und fehleranfällig macht. Die zwei Kurven darunter zeigen Finanzmarktkurven (links) und die dazugehörige Time-Wrapping Funktion (rechts, Sakoe und Chiba 1978). Eine solche Funktion kann dazu verwendet werden, zwei Kurven anzugleichen. In diesem Fall erkennt man sehr deutlich zwei starke Nichtlinearitäten, die sowohl eine Phasen- als auch eine Frequenzverschiebung andeuten. Dies muss in Betracht gezogen werden, wenn man die Kurven vergleichen möchte.

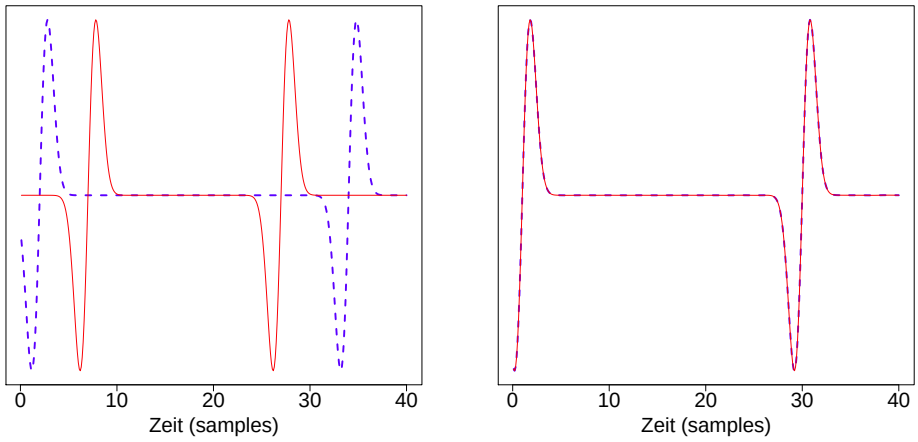


Abbildung 4.4: Die Kurven auf der linken Seite zeigen die Abstraktion zweier Herzschläge. Deutlich erkennbar ist dabei der zeitliche Unterschied zwischen dem ersten und dem zweitern Herzschlag. Nach einer Abbildung dieser Daten auf einen RDS (und anschließender Rekonstruktion) erhalten wir zwei perfekt abgestimmte Herzschläge (sichtbar in der Kurve auf der rechten Seite). Der Grund hierfür liegt in der Ausrichtung der Herzschläge an den Wörterbuchelementen. Durch die Berechnung des Skalarprodukts, mit Hilfe des Dynamic-Time-Wrapping Kernels, zwischen den Herzschlägen und den Wörterbuchelementen findet eine Ausrichtung der Schläge an den Wörterbuchelementen, statt. Diese Ausrichtung ist für alle Datenelemente gleich und wir erhalten, auch für beliebige Daten, perfekt ausgerichtete Zeitreihen.

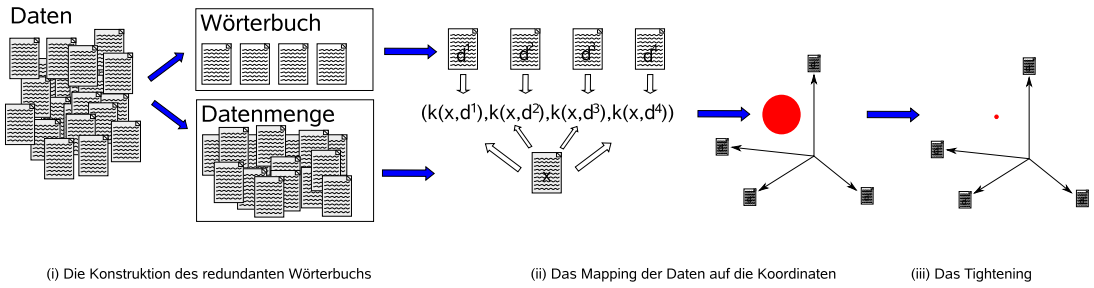


Abbildung 4.5: Die drei Schritte (Wörterbucherstellung, Projektion und Tightening) zur Projektion vektorieller und nicht vektorieller Daten auf Vektorräume, basierend auf redundanten Wörterbüchern.

KAPITEL

5 PRAXIS DER KOMPRESSIONSBASIERTEN MUSTERERKENNUNG

Überblick

- Mit Hilfe der kompressionsbasierten Mustererkennung lassen sich die Ergebnisse der diskreten NCD Experimente reproduzieren (Genomgruppierung in Absch. 5.1.1 und Autorengruppierung in Absch. 5.1.2).
- Kompressionsbasierte Mustererkennung lässt sich auch in Verfahren wie Support-Vector-Machinen einbauen. Klassifikationsexperimente mit kompressionsbasiertem Mercer-Kernel in Absch. 5.2.2.
- Bei der Bild- (Absch. 5.2.4) und Texturklassifikation (Absch. 5.2.5) mit Hilfe kompressionsbasierter Verfahren lassen sich gute Ergebnisse erzielen.

Nachdem wir uns im vorangegangenen Kapitel die Theorie der kompressionsbasierten Mustererkennung angesehen haben, werden wir uns in diesem Kapitel der Praxis widmen. Hier wird es nun darum gehen, wie erfolgreich die beschriebenen Methoden sind: Einerseits im Vergleich zu bestehenden aber auch auf neue Fragestellungen angewendet. Wir werden uns deshalb zuerst der diskreten Mustererkennung zuwenden. Dabei vergleichen wir, inwieweit sich mit der kompressionsbasierten Mustererkennung die Ergebnisse der NCD reproduzieren lassen. Danach werden wir uns der kontinuierliche Mustererkennung zuwenden. Hier geht es darum, klassische Mustererkennungsaufgaben mit Hilfe kompressionsbasierter Methoden zu lösen.

5.1 Diskrete Mustererkennung

In diesem Abschnitt werden wir uns die bekannten Experimente aus den NCD Papers von Cilibrasi und Vitànyi sowie Li et al. ansehen. Wir werden dabei jeweils das NCD Experiment rekonstruieren und die Ergebnisse mit denen vergleichen, die wir mit kompressionsbasierten Verfahren gewonnen haben.

5.1.1 Genomgruppierung

Die NCD wurde schon auf zahlreiche Bereiche der Mustererkennung angewendet, wir werden uns hier die Gruppierung von Säugetieren ansehen. Dieses Experiment wurde von Cilibrasi und Vitànyi (2005) und Li et al. (2001) beschrieben und zielt darauf ab, Säugetiere anhand ihrer DNA in Gruppen einzuteilen.

Die Hauptaufgabe bei diesem Experiment ist es, einen phylogenetischen Baum der Säugetiere zu erstellen. Dabei wird die gesamte mitochondriale DNA der beteiligten Tiere herangezogen. Das ursprüngliche Ziel war, eine Theorie über die Verwandtschaftsbeziehungen der drei Gruppen (Nagetiere, Primaten und Ferungulata) zu bestätigen. Hierfür verwendeten Li et al. 20 Säugetier mtDNA Sequenzen aus den vier verschiedenen Gruppen (die drei bereits erwähnten und eine weitere sogenannte Outgroup), um die von Cao et al. (1998) beschriebenen Verwandtschaftsverhältnissen zu untersuchen. Wir werden uns jedoch nicht um diesen biologischen Hintergrund kümmern, sondern ausschließlich den Vergleich der Mustererkennungsverfahren in den Vordergrund stellen.

NCD Experiment

Das Originalexperiment von Li et al. vergleicht die Ähnlichkeit von 20 Spezies mit Hilfe der NCD Formel. Der dabei verwendete Kompressor ist GenCompress (Chen et al. 2000). Die Distanzmatrix – die durch den Vergleich jeder der 20 Spezies mit jeder anderen berechnet wird – dient dem *neighbor-joining* Verfahren aus dem ProtML Paket[1] als Basis für die Gruppierung. Durch den Einsatz dieser Methode lassen sich Schwierigkeiten im Bereich statistischer Einflüsse umgehen, die mit den üblichen Methoden auftreten würden (Husmeier et al. 2004). Ein anderes Experiment von Li et al. arbeitet mit 24 Spezies und verwendet zur Gruppierung die Quarted Methode von Cilibrasi und Vitànyi (2005).

Kompressionsbasiertes Experiment

Im Rahmen unseres Experiments werden wir versuchen, die identischen Daten mit Hilfe der im vorherigen Kapitel beschriebenen, kompressionsbasierten Methode zu

[1] Das ProtML Paket von Jun Adachi und Masami Hasegawa zur Berechnung von evolutionsbiologischen Bäumen `http://cmgm.stanford.edu/phylip/protml.html`

vergleichen und anschließend mit der neighbor-joining Methode gruppieren. Das Ergebnis kann man in Abbildung 5.1 sehen. Es ist identisch mit dem Baum, den man erhält, wenn man NCD mit *bzip2* verwendet.

Der genaue Versuchsaufbau ist wie folgt: Wir teilen die mtDNA Sequenzen in Blöcke identischer Größe. Diese dienen als Wörterbuchelemente für die redundante Repräsentation. Eine mtDNA Sequenz hat dann eine binäre Sequenzrepräsentation, bestehend aus Nullen und Einsen. Für jeden Block, der in dieser Weise in der Sequenz vorkommt, steht eine Eins, für jeden, der nicht vorkommt, eine Null. In das innere Produkt (beschrieben in Definition 4.2) fließen also nur die Ähnlichkeiten zwischen den Wörterbucheinträgen ein. Das Block-Wörterbuch ist natürlich ein sehr grobes und dessen Wahl beliebig, aber es erfüllt die Anforderungen an ein Wörterbuch und ist mit den Definitionen des kompressionsbasierten Verfahrens verträglich. Als Blockgröße werden wir 300 Zeichen (mit dieser Größe erhielten wir die besten Ergebnisse) annehmen und die Blöcke überlappen sich nicht. Die exakte Berechnungsvorschrift für zwei Sequenzen f_1 und f_2 ist wie folgt:

1. Teilen Sie f_1 und f_2 in 300 Zeichen lange Blöcke, speichern Sie dies als Wörterbücher D_1 und D_2.

$$\mathcal{D}_{\#} = \{d | d = f_{\#}\left([i \cdot 300, (i+1) \cdot 300]\right), \\ i \in \{0, ..., n_{\#}/300\} \text{ wobei } \# \in \{1, 2\}\}$$

   Dieser Schritt generiert auch gleich das *virtuelle* Wörterbuch $\mathcal{D}$, das als Basis für die Kompression dient. Auf diese Weise erhalten wir zu jeden Wörterbuchelement die Anzahl und die Position und speichern sie in der Sequenzrepräsentation

$$C_{\mathcal{D}}(f_{\#}) = x_{\#}$$

2. Berechnen Sie das innere Produkt mit Hilfe der Formel aus Definition 4.2, d.h. vergleichen Sie jedes Element aus D_1 mit jedem anderen Element aus D_2 und gewichten Sie es mit der Frequenz mit der es in der DNA-Sequenz vorkommt.

$$\langle f_1, f_2 \rangle_{C_{\mathcal{D}}} = \sum_{d_1 \in D_1} \sum_{d_2 \in D_2} \langle d_1, d_2 \rangle x_1^{d_1} \cdot x_2^{d_2}$$

   Die Ähnlichkeit $\langle d_1, d_2 \rangle = l_{1,2}$ berechen wir dabei mit Hilfe von $l_{1,2}$, der Länge der längsten gemeinsamen Zeichenkette aus d_1 und d_2.

3. Normalisieren Sie die Ergebnisse

$$\frac{\langle f_1, f_2 \rangle_{C_{\mathcal{D}}}}{\sqrt{\langle f_1, f_1 \rangle_{C_{\mathcal{D}}}} \cdot \sqrt{\langle f_2, f_2 \rangle_{C_{\mathcal{D}}}}}$$

Die hiermit erhaltenen Ähnlichkeiten können nun in Distanzen überführt werden und als Eingabe für ein Clusteringverfahren dienen. Die somit erhaltene Gruppierung kann man in Abbildung 5.1 sehen.

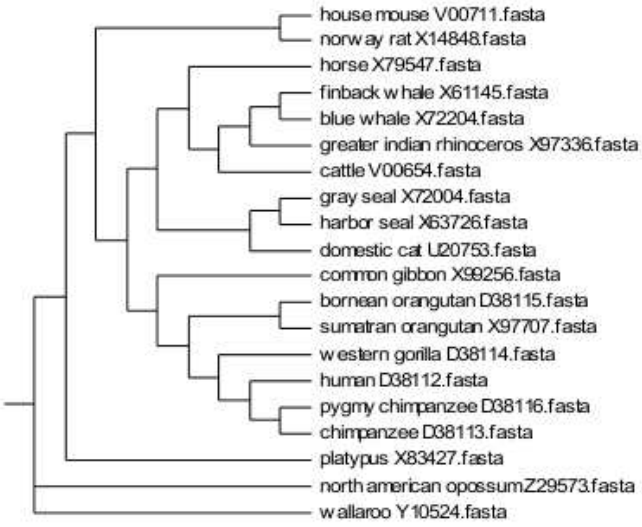


Abbildung 5.1: Hierarchisches Clustering der Säugetiere mit Hilfe des kompressionsbasierten Ansatzes aus Abschnitt 5.1.1.

5.1.2 Gruppierung russischer Autoren

Ein weiteres beliebtes Beispiel für die Fähigkeiten der NCD ist die Gruppierung russischer Autoren anhand der Ähnlichkeiten ihrer Werke.

Versuchsaufbau

Die Idee der Autorengruppierung wurde von Cilibrasi und Vitànyi in ihrer Arbeit „Clustering by Compression“ vorgestellt. Dabei geht es darum, Texte anhand ihrer Ähnlichkeit in Gruppen einzuteilen. Im Idealfall werden alle Texte eines Autors in einer Gruppe zusammengefasst. Das Verfahren „erkennt“ dann sozusagen die Autorenschaft. In dem Experiment von Cilibrasi und Vitànyi werden 19 klassische russische Texte gruppiert. Dies sind die Bücher von I.S. Turgenew (Am Vorabend, Rudin, Väter und Söhne, Ein Adelsnest), F.M. Dostoyevsky (Arme Leute, Der Idiot, Der Spieler, Schuld und Sühne), L.N. Tolstoi (Kindheit, Anna Karenina, Die Kosaken, Krieg und Frieden), N.W. Gogol (Geschichte des Streitfalls Iwan Iwanowitsch gegen Iwan Nikiforowitsch, Taras Bulba, Die toten Seelen, Das Portrait) und M.A. Bulgakow (Der Meister und Margarita, Hundeherz, Die verhängnisvollen Eier). Die Werke sind frei in Bibliotheken im Internet verfügbar (`http://www.lib.ru/`, `http://www.gutenberg.org/`). Intuitiv würde man erwarten, dass Texte des selben Autors in einer Gruppe zusammengefasst werden, was wir auch später sehen werden. Eine mögliche praktische Anwendung dieses Verfahrens wäre zum Beispiel die Erkennung von Plagiaten.

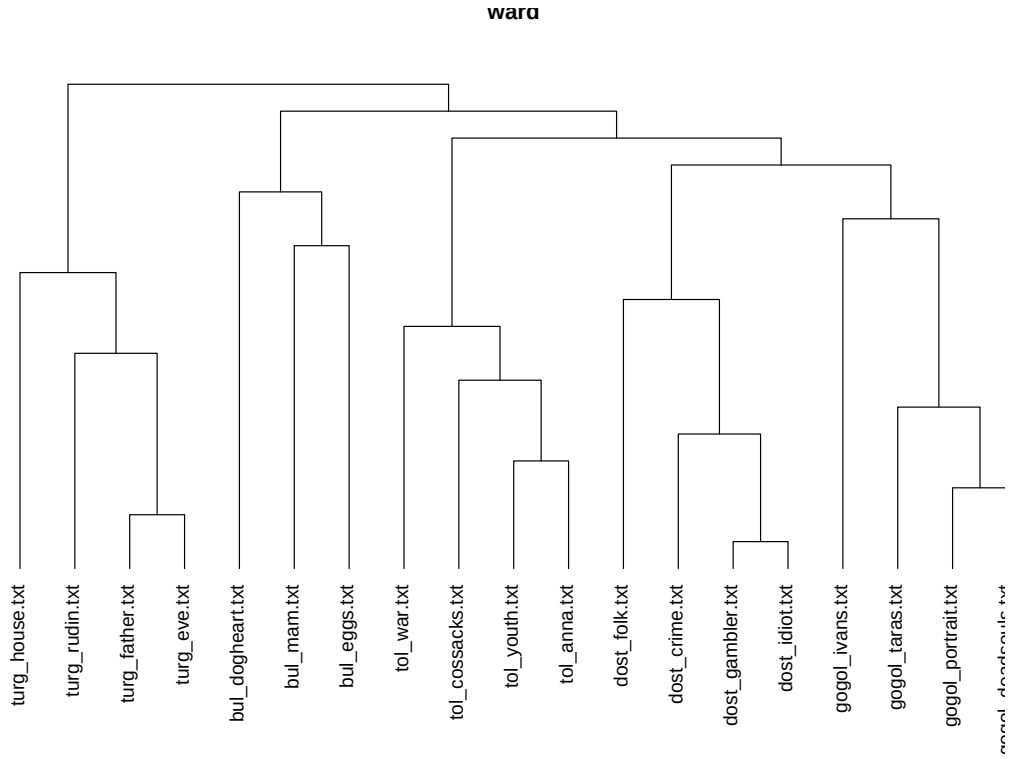


Abbildung 5.2: Hierarchisches Clustering der russischen Autoren mit Hilfe des kompressionsbasierten Ansatzes aus Abschnitt 5.1.2.

NCD Experiment

Um die Ergebnisse von Cilibrasi und Vitànyi nachvollziehen zu können, haben wir ihren Versuchsaufbau kopiert. Die exakte Beschreibung dieses Experiments haben wir schon in Kapitel 3.2.2 behandelt, daher hier nur eine kurze Zusammenfassung.

Cilibrasi und Vitànyi nutzen für ihr Experiment den *bzip2* Kompressor und die NCD, um die Distanzmatrix zu bestimmen. Dies ist aufgrund der größeren Puffergröße auch eine gute Wahl. Wir haben jedoch in Abschnitt *Autorengruppierung* (Kapitel 3.2.2) bereits demonstriert, dass LZ77 mit größerem Puffer sehr wohl in der Lage ist, identische Ergebnisse zu erzielen. Um die Daten basierend auf der Distanzmatrix zu gruppieren, verwenden Cilibrasi und Vitànyi die Quarted Clustering Methode. Das Ergebnis war eine perfekte Gruppierung der Autoren.

Kompressionsbasiertes Experiment

Hier nun der Aufbau des Experiments für das kompressionsbasierte Verfahren. Aufgrund der großen Dateigröße werden wir nur die ersten $70kB$ der Dokumente betrachten, was sich als ausreichend erweisen wird. Nachdem das kompressionsbasierte Ver-

fahren die Ähnlichkeiten bestimmt hat, werden die Daten mit dem Wart-linkage Clusteringverfahren (Ripley 2007) gruppiert. In einigen Fällen war die Dokumentengröße kleiner als die maximal möglichen $70kB$. In diesen Fällen betrachten wir die Dokumente in ihrer Originalgröße.

Die eigentliche kompressionsbasierte Ähnlichkeitsberechnung ist identisch mit der des mtDNA Experiments (aus Abschnitt 5.1.1) mit dem Unterschied, dass die Blockgröße in diesem Fall 1.700 Bytes beträgt.

Abbildung 5.1.2 zeigt das Ergebnis der Gruppierung. Diesen Baum erhalten wir für jeden der beschriebenen Ansätze (kompressions- und NCD-basiert). Wir können damit also zeigen, dass für den Fall der russischen Autoren und der mtDNA identische Ergebnisse erzielbar sind. Dies sagt natürlich noch nichts über andere Experimente aus, es legt jedoch, zusammen mit der methodischen Ähnlichkeit der Verfahren, nahe, dass auch andere Experimente ähnlich ausgehen werden.

5.2 Kontinuierliche Mustererkennung

Wir werden uns nun der kontinuierlichen Mustererkennung zuwenden. Dabei betrachten wir zuerst einmal einen, was die Überprüfung der Ergebnisse angeht, sehr einfachen Fall: die Klassifikation von Phonemen (zu finden in Hastie et al. 2002; Ferraty und Vieu 2006). Wir werden hier ausschließlich mit der vom Umfang her kürzeren Variante von Ferraty und Vieu arbeiten.

Wir werden im Folgenden nicht die reine Ähnlichkeit zum Vergleich heranziehen, sondern uns auf Support Vector Machines (abgekürzt SVM, Vapnik 1998) stützen. Konkret bedeutet dies, dass wir das innere Produkt (basieren auf redundanten Wörterbüchern) als Kern-Funktion in einer SVM benutzen werden. Dies ermöglicht leistungsfähigere Klassifikatoren und demonstriert gleichzeitig die allgemeine Verwendbarkeit kompressionsbasierter Mustererkennung.

5.2.1 Wörterbuchberechnung

Bisher sind wir immer davon ausgegangen, das Wörterbuch existiere bereits. Hier müssen wir uns nun explizit um ein Wörterbuch bemühen. Für diesen Anwendungsfall werden wir ein sehr einfaches Wörterbuch berechnen. Unser Ansatz wird dabei das sogenannte Random Sampling, also das zufällige Auswählen, sein. Dies wird von Bishop (2006) ausführlich beschrieben. Es geht dabei darum, eine Menge von Funktionen E_i aus einer Beispielmenge wie folgt zu bestimmen

$$E_i = \sum_{j}^{N} f_{k_j^i} \cdot P(k_j^i).$$

Hierbei wird $E_i \in \mathcal{H}$ als die gewichtete Summe von Beispielen $f_{k_j^i}$ aus einem Trainingsset $\mathcal{T} = \{f_k\}_{k \in \mathbf{T}}$ mit Indexmenege $|\mathbf{T}| < \infty$ betrachten Die Verteilungsfunktion

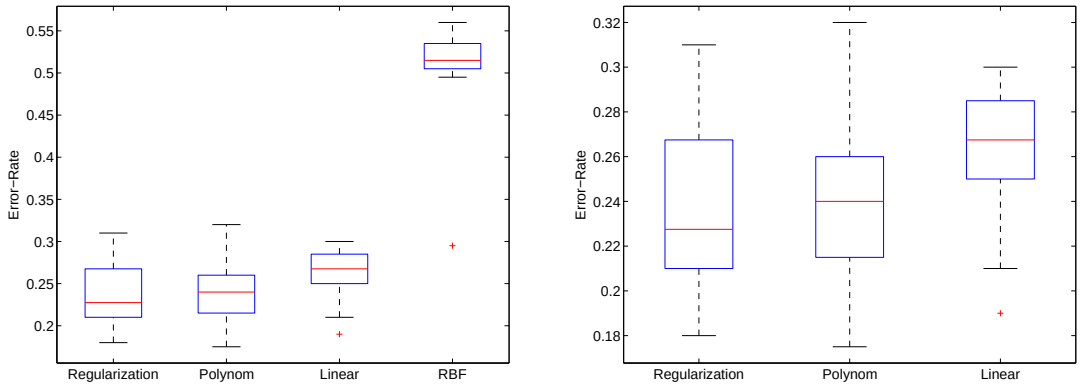


Abbildung 5.3: Die Ergebnisse der Phonemeklassifikation: Der kompressionsbasierte Ansatz (der Regularization Kernel, hier im Zusammespiel mit Support Vector Machines) arbeitet besser als andere Kerne. Der Vergleich mit Polynomkernen ist zwar nicht statistisch signifikant, was jedoch in diesem Kontext durchaus vertretbar ist, da sich das log-Spektrum der Phoneme ideal mit Polynomen approximieren lässt.

$P(k_j^i)$ bestimmt dabei den Einfluss, den ein Beispiel auf die Funktion hat, wobei wir der Einfachheit halber Gleichverteilung annehmen werden. Die k_j^i werden für alle i zufällig bestimmt.

Mit Hilfe dieser Vorgehensweise erhalten wir ein Wörterbuch der Form $\mathcal{D} = \{E_i\}$ wobei $i \in \{1...n\}$ ist. Die einzelnen Einträge werden wie folgt berechnet:

$$E_i = \frac{1}{N} \sum_{j=1}^{N} f_{k_j^i} \tag{5.1}$$

dabei wird der Index k_j^i willkürlich aus der Indexmenge bestimmt $\mathbf{T}$. Nach Bishop genügt ein N der Größe zehn, um aufbauend auf den unabhängigen Beispieldaten eine gute Schätzung für E_i abzugeben.

Bei Klassifikationsproblemen sollte das Wörterbuch so gestaltet sein, dass es Elemente gibt, die sowohl einzelnen Klassen als auch klassenübergreifende Aspekte repräsentieren. Dies kann man durch eine bewusste Auswahl der einzelnen Funktionen erreichen, oder indem man die Anzahl N, der in Betracht zu ziehenden Elemente sehr klein wählt.

5.2.2 Phonemeklassifikation

Der Phonemedatensatz besteht aus 2.000 Datenelementen die jeweils einer der fünf Klassen entstammen. Jedes Phonem ist als Log-Periodogram repräsentiert. Wir werden uns in diesem Experiment jedoch nur um die Log-Periodogramme zu 'aa', wie in dem englischen 'dark' und 'ao', wie in 'Water' kümmern. Von diesen gibt es insgesamt 600 Beispiele. Wie man in Abbildung 5.4 sehen kann, gibt es zwischen den mittleren Kurven nur sehr subtile Unterschiede. Diese verschwinden jedoch völlig wenn man

alle Kurven betrachtet. Um einen realistischen Eindruck für die Leistungsfähigkeit kompressionsbasierter Verfahren zu bekommen, werden wir diese mit anderen SVM-basierten Verfahren vergleichen. Dabei werden wir gängige SVM-Kerne einsetzen, um einen realistischen Vergleich zu erhalten.

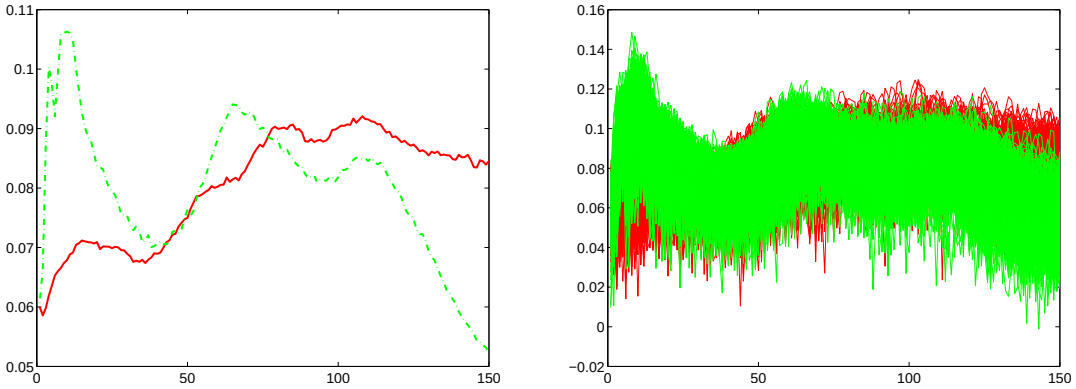


Abbildung 5.4: Die beiden Diagramme stellen die mittleren Kurven (links) und alle Kurven (rechts) der Phoneme ’aa’ (rot) und ’ao’ (grün) dar. Dabei fällt besonders auf, dass sich die mittleren Kurven (der Mittelwert über alle Kurven) sehr deutlich unterscheiden, dass die Wertschwankungen für die Gesamtheit der Kurven jedoch erheblich ist. Im rechten Diagramm ist es bei Weitem nicht mehr so einfach die unterschiedlichen Merkmale (wie zum Beispiel positiv verlaufenden Steigung bei den roten Kurven oder die zwei Peaks im hinteren Bereich der grünen Kurven) der Kurven auszumachen wie links.

Unsere Experimente sehen wie folgt aus. Aus den 600 zur Verfügung stehenden Beispielen ziehen wir 200 unterschiedliche zur Wörterbuchgenerierung heran. Wir nutzen diese also, um mit Hilfe von Formel (5.1), 200 Wörterbuchelemente zu generieren. Die verbleibenden 400 Beispiele werden willkürlich in 200 Test- und 200 Trainingsbeispiele unterteilt. Mit Hilfe der Trainingsdaten wird die Support Vector Machine trainiert. Dabei trainieren wir unterschiedliche Kernfunktionen. Zum Vergleich ziehen wir Kernfunktionen basierend auf *linearen*, *radialen* und *polynomiellen* Basisfunktionen heran. Für die kompressionsbasierten Verfahren nutzen wir das innere Produkt auf redundanten Wörterbüchern als Mercerkernel. Nach dem Training berechnen wir dann den Klassifikationsfehler E auf den Testdaten.

$$E = \frac{1}{200} \sum_{f \in \text{Test}} \mathbf{1}_{\{\text{SVM}(f) \neq c_f\}}$$

Wobei $\text{SVM}(f)$ das Ergebnis der Support Vector Machine Klassifikation ist und c_f die tatsächliche Klassenzugehörigkeit der Funktion f. Die Ergebnisse zeigen eine Verbesserung gegenüber den Ergebnissen von Ferraty und Vieu (2006) und sind vergleichbar mit denen von Hastie et al. (2002). Eine kurze Zusammenfassung ist in Tabelle 5.2.2 zu sehen, visuell sind sie auch noch in Abbildung 5.3 dargestellt.

	$\mathrm{SVM}_{\mathrm{comp}}$	$\mathrm{SVM}_{\mathrm{poly}}$	$\mathrm{SVM}_{\mathrm{linear}}$	$\mathrm{SVM}_{\mathrm{rbf}}$
mean	0.21	0.25	0.27	0.51
min	0.15	0.17	0.25	0.50

Tabelle 5.1: Fehlerraten der Phonemeklassifikation

5.2.3 Verrauschte Phonemeklassifikation

Die gerade beschriebene Phonemeklassifikation ist ein schwieriger Test. Wir fügen noch etwas Rauschen zu den Daten hinzu, um den Grad der Schwierigkeit weiter zu erhöhen. Dabei unterscheiden wir zwischen sogenanntem *white noise*[2], das ist normalverteiltes Rauschen mit Mittelwert 0 und konstanter Varianz, und anders geartetem Rauschen. Das nichtstationäre, also nicht weiße Rauschen implementieren wir mit Hilfe der Sinc Funktion s die um den Faktor t verschoben ist

$$s_t(x) = \frac{sin(\pi(x-t))}{\pi(x-t)}.$$

Des Weiteren addieren wir noch weißes Rauschen

$$w(x) = X(t) \quad \text{mit} \quad X(x) \text{ ist } N(0,1) - \text{verteilt}$$

auf diese Funktion. Dieses Rauschen (weißes Rauschen und Sinc-Funktion) addieren wir mit der Gewichtung $1/10$ auf das jeweilige Phonemebeispiel. Das Rauschen ist mit $1/4$ weißem Rauschen und $3/4$ Sinc verteilt. Der Einfluss dieses Rauschens kann in Abbildung 5.5 und Abbildung 5.6 wahrgenommen werden. Dabei kann man einerseits sehen wie sich das Rauschen zusammensetzt und andererseits die "saubere„ und die verrauschte Kurve sehen. Das Experiment selbst ist identisch mit dem Vorangegangenen.

	$\mathrm{SVM}_{\mathrm{comp}}$	$\mathrm{SVM}_{\mathrm{poly}}$	$\mathrm{SVM}_{\mathrm{linear}}$	$\mathrm{SVM}_{\mathrm{rbf}}$
mean	0.26	0.25	0.34	0.51
min	0.18	0.18	0.28	0.47
std	0.04	0.03	0.03	0.02

Tabelle 5.2: Fehlerraten der Phonemeklassifikation

Die Ergebnisse zeigen eine starke Resistenz des polynomiellen Kernels gegenüber Rauschen. Dies reduziert den Vorsprung des redundanten Wörterbuchs der sich nicht so gut an die neuen Gegebenheiten anpassen kann wie der Polynomkernel. Verglichen mit den anderen Kernen arbeitet der Redundante-Wörterbuch-Kern jedoch immer noch am effektivsten. Eine mögliche Interprätation ist, dass sich die Polynome sehr gut an die Struktur der Daten anpassen können und dabei Rauschen herausfiltern. Dies ist mit

[2]Eine schöne Eigenschaft des weißen Rauschens ist die Tatsache, dass man es durch Mittelung entfernen kann.

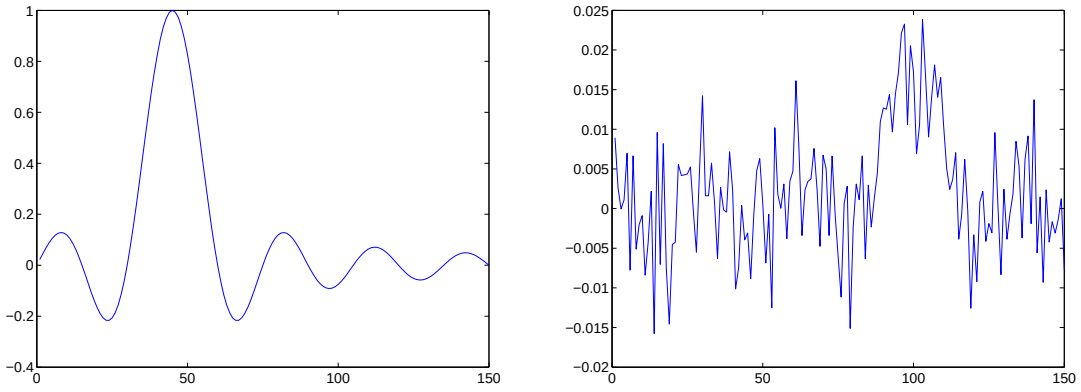


Abbildung 5.5: Sinc-Funktion und Sinc-Funktion mit Rauschen. Diese Funktionen werden, mit variierenden Parametern, auf die Daten aufaddiert, um nichtstationäres Rauschen zu simulieren.

dem redundanten Wörterbuch nur bedingt möglich, da weniger Hintergrundwissen in die Klassifikation mit einfließt. Dies kann man auch daran erkennen, dass Hastie et al. (2002) als ideale Basis stückweise polynomielle Funktionen heranzieht. Mit zunehmendem Rauschen konvergieren jedoch alle Kernfunktionen zu einem konstanten (und identischen) Wert (eine Fehlerrate von 0.5 entspricht einfachem Raten).

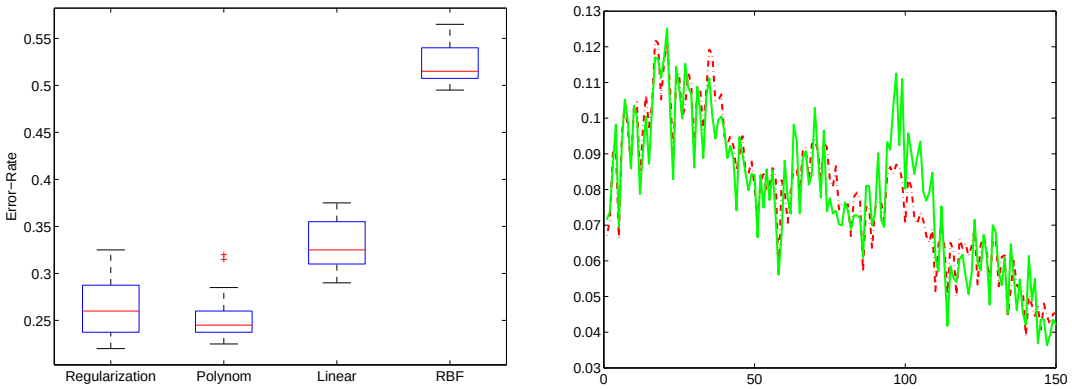


Abbildung 5.6: Ergebnisse zu den verrauschten Daten (links) und der Unterschied zwischen verrauschtem (durchgezogene grüne Linie) und sauberem (gepunktete rote Linie) Signal. Der Vorsprung des Regularisierungskernels wurde vom Polynomkernel in diesem Experiment reduziert und der Polynomkernel liefert sogar etwas bessere Ergebnisse. Dies liegt vermutlich daran, dass Polynome eher der Struktur der Daten entsprechen. Dieser „Vorteil“ ermöglicht es ihnen, besser zu klassifizieren als dem Regularisierungskernel, der ausschließlich mit dem Wissen, das er aus den Daten zieht, arbeitet.

Nachdem wir uns einfache $1D$ Funktionen angesehen und dabei auch gut zu interpretierende Ergebnisse bekommen haben, werden wir nun $2D$ Bild-Daten betrachten. Damit werden wir nicht nur die Rechenkomplexität steigern, sondern auch noch die

Realitätsnähe erhöhen. Konkret bedeutet dies, dass wir uns zwei Szenarien ansehen werden. Einerseits Texturklassifikation und andererseits Szenenklassifikation.

5.2.4 Bildklassifikation

Das Experiment zur Klassifikation natürlicher Bilder besteht aus drei Datensätzen mit Bildern der Corel Image CDs. Die drei Sätze sind Militärflugzeuge, Propellerflugzeuge und Delfine. Jeder Datensatz besteht aus 100 Bildern. Für unser Experiment wählen wir 100 aus diesen 300 Bildern aus und klassifizieren sie mit Hilfe des k-Nächste-Nachbarn Verfahren. Dazu wählen wir aus den 100 Bildern zufällig ein Bild aus. Aus den verbleibenden 99 wählen wir dann noch die $k = 7$ nächsten Nachbarn zu diesem. Diese 7 Bilder dienen uns zur Entscheidung: Die darin am häufigsten vertretene Klasse weisen wir dem ausgewählten Bild zu und überprüfen unsere Auswahl. Um die Rechenzeit zu reduzieren, führen wir diese Berechnungen nicht auf den Originalbildern, sondern auf einer auf 100×100 Pixel heruntergerechneten Version durch.

Abbildung 5.7: Eine zufällige Auswahl an Bildern aus dem Bildklassifikationsexperiment.

Die Wahl für kNN als Klassifikationsmethode liegt darin begründet, dass es sich dabei um eine sehr einfache und fast ausschließlich von dem gewählten Distanzmaß abhängende Methode handelt. Es sollte unser Ziel sein, den Einfluss der Klassifikationsmethode auf das Ergebnis so gering wie möglich zu halten. Das Nächste-Nachbarn Verfahren kommt diesem Anspruch sehr nahe, da es wie Ripley (2007) schon bestätigt, in kritischem Maß von dem gewählten Abstandsmaß abhängt. Die Anzahl der nächsten Nachbarn $k = 7$ ist eine Abwägung zwischen Abstandsmaß und Rauschen. Je kleiner

die Zahl, umso mehr verlässt man sich auf das Abstandsmaß, je größer, umso stärker der Fokus auf die Beispieldaten. Alles in allem werden wir zwar den Fokus auf das Distanzmaß legen, jedoch ein realistisches Bild der Klassifikationsaufgabe wiedergeben. Aus diesem Grund stellt $k = 7$ einen guten Kompromiss zwischen der Komplexität der Aufgabe und dem Einfluss des Distanzmaßes dar.

In diesem ersten Experiment werden wir die kompressionsbasierte Methode mit der euklidischen Distanz und der Normalisierten Kompressionsdistanz vergleichen (NCD).

Kompressionsbasiertes Abstandsmaß Kompressionsbasierte Verfahren basieren zum großen Teil auf der Wahl des zugrunde liegenden Wörterbuches. Da es jedoch unendlich viele Möglichkeiten gibt, fällt die Wahl sehr schwer. Wir werden uns daher mit sehr einfachen Wörterbüchern begnügen. Wir wählen einfach eine zufällige Teilmenge der Daten als Wörterbuch aus. Diese dürfen dann natürlich nicht mehr bei der Klassifikation in Betracht gezogen werden. Wir teilen also den Datensatz (300 Bilder) in 100 Testbilder, die mit dem kNN Verfahren verwendet werden, und 200 Wörterbuchbilder für die Kompression mit Hilfe dieser Bilder. Zu jedem Bild berechnen wir dann die komprimierte Repräsentation bezüglich des Wörterbuchs. Hierfür bieten sich Matching Pursuit (Abschnitt 4.1.1) oder Basis Pursuit (Abschnitt 4.1.2) an. Wir werden Matching Pursuit verwenden. Nun können wir die komprimierte Repräsentation für den Vergleich der Bilder heranziehen, dazu verwenden wir das innere Produkt bezüglich redundaten Wörterbüchern und transformieren das Ergebnis in eine Distanz. Aufbauend hierauf können wir mit Hilfe des kNN Verfahrens die Klassenzugehörigkeit der Bilder bestimmen.

Euklidische Distanz Das Vorgehen in diesem Fall ist sozusagen die Urversion des kNN. Die Ähnlichkeit bestimmen wir hier einfach mit der euklidischen Distanz, wie sie in den meisten Büchern über Mustererkennung vorgestellt wird (z.B. Hastie et al. 2002).

NCD Die Anwendung der NCD auf den, unkomprimierten BMP[3] Bildern ist sehr geradlinig. Wenn die Bilder einmal konkateniert und komprimiert sind, muss nur noch die Dateigröße in die NCD Formel eingegeben werden. Zu diesem Zweck erstellen wir kleinere (100×100 Pixel) Varianten der Originalbilder, konkatenieren diese mit dem frei verfügbaren ImageMagick Toolkit und komprimieren sie mit dem *bzip2* Kompressor. Die Berechnung der NCD und das kNN Verfahren finden dann vollständig in MATLAB statt. Das kNN Verfahren ist identisch mit allen anderen hier beschriebenen Vorgehensweisen.

[3]BMP Bilder werden von den meisten Programmen, unter anderem MATLAB, als unkomprimierte Pixelfolge codiert, was die Anwendung der NCD auf den rohen Bilddaten ermöglicht

Ergebnisse Da ein Ergebnis natürlich maßgeblich von der Wahl der 100 Testbilder abhängt, wiederholen wir die Tests 10 mal und präsentieren die Ergebnisse in Form von Boxdiagrammen mit Median, unterem und oberen Quartil. Die Box selbst gibt die Werte innerhalb des 1.5-fachen Interquartilbereichs an (weitere Informationen finden sich in der MATLAB Dokumentation Matlab 2008). Abbildung 5.8 zeigt die dabei gewonnenen Ergebnisse. Wie zu sehen ist, liefern sowohl euklidische als auch kompressionsbasierte Distanzen bessere Ergebnisse als die NCD. Der Unterschied zwischen den beiden selbst ist jedoch eher gering. Dies mag an der einfachen Struktur des Wörterbuchs liegen. Wenn man das alles in Betracht zieht, so sind die Ergebnisse jedoch sehr vielversprechend.

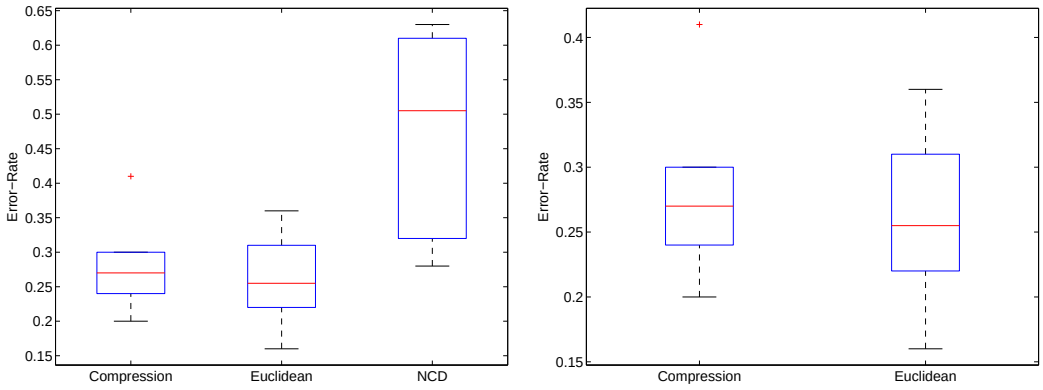


Abbildung 5.8: Fehlerraten für die Klassifikation natürlicher Bilder der unterschiedlichen Methoden. Beide Methoden, Kompression und euklidischer Abstand, liefern bessere Ergebnisse als die NCD.

5.2.5 Texturbilderklassifikation

Das Texturklassifikationsexperiment basiert auf Bilddaten aus der Brodatz-Texturdatenbank[4] (Brodatz 1966). Sie besteht aus 512×512 Pixel Grauwertbildern (256 Grauwerte) von verschiedenen Texturen. Für dieses Experiment werden wir jedoch nur die ersten sieben Texturen heranziehen. Die genauen Typen sind in Tabelle 5.3 aufgelistet. Von jeder Textur werden wir 16 überlappende 200×200 Pixel Ausschnitte berechnen und diese jeweils auf eine Größe von 100×100 Pixel herunterrechnen. Als Ergebnis erhalten wir 112 Texturen, die als Datenbasis benutzt werden können. Neben diesem Datensatz betrachten wir auch noch einen Datensatz mit rotierten Texturen. Diese stammen vom selben Datensatz (den einhundertzwölf 100×100 Bilder) ab.

Um die Daten zu klassifizieren, betrachten wir jedoch nicht die Originaldaten, sondern die fouriertransformierten Ausschnitte. Diese werden mit einem k-Nächste

[4]Die Daten stammen von der USC Homepage (`http://sipi.usc.edu/database/database.cgi?volume=textures`).

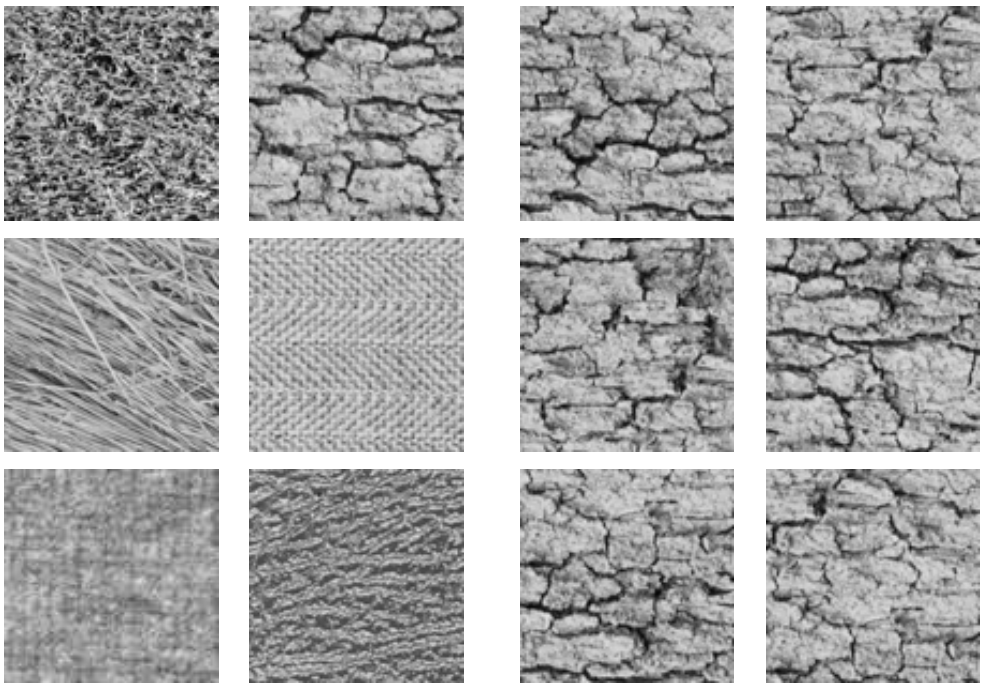


Abbildung 5.9: Eine Auswahl an Bildern aus der Brodatz-Texturdatenbank, links sind Beispiele der unterschiedlichen Texturen und rechts sind die Flicken einer Textur.

Nachbarn Ansatz klassifiziert. Konkret bedeutet dies, dass wir die annotierten Daten – zu jeden Bild ist die Klassenzugehörigkeit bekannt – fouriertransformieren und das kNN Verfahren in der Fourierdomäne anwenden. Um die Handhabung komplexer Werte zu erleichtern, betrachten wir nur den Realteil der transformierten Daten. Für jeden Ausschnitt suchen wir dann die $k = 7$ nächsten Nachbarn und bestimmen die Klassenzugehörigkeit des Ausschnitts anhand der häufigsten Klasse der Nachbarn. Die Abstandsberechnung ist dabei von dem gewählten Verfahren abhängig:

1. Die kompressionsbasierte Ähnlichkeitsberechnung mit Hilfe des inneren Produkts über redundanten Wörterbüchern und der sog. Bump-Algebra als Wörterbuch.

Texture	B. page number
Grass	(D9)
Bark	(D12)
Straw	(D15)
Herringbone weave	(D15)
Woolen cloth	(D19)
Pressed calf leather	(D24)
Beach sand	(D29)

Tabelle 5.3: Texturklassen

2. Die kompressionsbasierte Ähnlichkeitsberechnung mit Hilfe des inneren Produkts über redundanten Wörterbüchern und der sog. Bump-Algebra sowie den Fouriertransformierten rotierten Bildern als Wörterbuch.

3. Die euklidische Distanz auf den fouriertransformierten Bildern.

4. Die NCD auf den fouriertransformierten Bildern.

Die Ergebnisse zu diesem Experiment kann man in Tabelle 5.4 und Abbildung 5.10 finden.

Kompressionsbasierter Ansatz mit Bump-Algebra-Wörterbuch Das Wörterbuch nimmt bei den kompressionsbasierten Ansätzen eine zentrale Rolle ein. Das bedeutet, dass man durch geschickte Ergänzug oder Bestimmung der Wörterbucheinträge die Qualität des Klassifikationsergebnisses verbessern kann. In diesem Experiment versuchen wir ein einfacheres Wörterbuch, die sogenannte Bump-Algebra (Meyer 1992). Diese besteht aus normalisierten Gaußglocken

$$g_{\mu,\sigma}(t) = \frac{1}{||g_{\mu,\sigma}||} \exp\left(-\frac{||x-\mu||^2}{2\sigma^2}\right)$$

und ist hochgradig redundant. Um den Rechenaufwand niedrig zu halten, betrachten wir g für eintausend μ und σ Kombinationen. Je nach Größe von $\sigma \in \{1, 5, 10, 50, 100\}$ wählen wir das Gitter, das durch die Werte von μ aufgespannt wird, feiner oder gröber. Da es sich bei diesem Wörterbuch um ein sehr Einfaches handelt, das auch unabhängig von der Fragestellung ist, können wir Aufschlüsse über dessen Einfluss auf das Verfahren erhalten. Für uns von Vorteil ist die Tatsache, dass Gaußglocken in der Fourierdomäne auch Gaußglocken sind.

Kompressionsbasierter Ansatz mit vollständigem Wörterbuch In diesem Experiment werden wir die Bump-Algebra noch um die rotierten Texturen ergänzen. Diese rotierten Ausschnitte werden auch wieder fouriertransformiert. Das bedeutet, dass das Wörterbuch besser an die Aufgabe angepasst ist und somit auch bessere Ergebnisse liefern sollte. Diese Auswahl entspricht dem Hinzunehmen von Hintergrundwissen.

Euklidische Distanz Die euklidische Distanz ist sozusagen die Standardversion eines Distanzmaßes. Sie wird in den meisten Büchern über Mustererkennung vorgestellt (z.B. Hastie et al. 2002) und besprochen. In diesem Experiment werden wir dieses Distanzmaß auf die fouriertransformierten Bilder anwenden.

Normalisierte Kompressionsdistanz Bei der Anwendung der NCD gibt es eigentlich nur einen Parameter, den man anpassen kann und das ist die Wahl des Kompressionsalgorithmuses. In den vorangegangenen Kapiteln sind wir sehr intensiv auf diese

Wahl eingegangen, hier werden wir uns jedoch hauptsächlich um den Vergleich mit den anderen Verfahren kümmern, weniger um den Vergleich der einzelnen Kompressionsmethoden. Daher wenden wir *bzip2* an, auch wenn dies keine detaillierte Analyse der Ergebnisse wie in den vorangegangenen Kapiteln erlaubt. Damit wir mit der NCD auch vergleichbare Ergebnisse erhalten, wenden wir sie auf die fouriertransformierten Bilder an. Diese werden als BMP Bilder abgespeichert, um keine Seiteneinflüsse von anderen Kompressionsverfahren (wie JPEG oder GIF) zu haben.

Methode	Mit. Fehler
Kontinuierliche Kompression (vollst.)	0.0446
NCD	0.1429
Kontinuierliche Kompression (Bump)	0.1696
Euklidisch	0.2232

Tabelle 5.4: Fehlerraten für die Texturklassifikation der unterschiedlichen Methoden

Der Versuchsaufbau ist eher einfach, wir nehmen je zwei Dateien, konkatenieren sie mit ImageMagick und komprimieren das resultierende Bild mit *bzip2*. Die damit erhaltene Dateigröße, sowie die der unkonkatenierten Dateien, dienen nun als Eingabe für die NCD. Die Anwendung des kNN Algorithmuses erfolgt nun analog zu den anderen Experimenten.

Ergebnisse Wie die Ergebnisse in Tabelle 5.4 und Abbildung 5.10 zeigen, liefern die kompressionsbasierten Ansätze deutlich bessere Ergebnisse als die NCD oder die euklidische Distanz. Reduziert man jedoch den Grad der Anpassung des Wörterbuchs, indem man nicht das vollständige Wörterbuch, sondern nur die Bump-Algebra verwendet, so ändert sich das Bild. Das kompressionsbasierte Verfahren ist zwar immer noch leistungsfähiger als die NCD, jedoch nicht mehr als die euklidische Distanz. Dies demonstriert auf der einen Seite die Stärke aber auch die Schwäche der kompressionsbasierten Verfahren. Ein gut adaptiertes Wörterbuch ist in der Lage, die Klassifikationsleistung deutlich zu steigern, ein schlecht adaptiertes leistet aber mindestens das selbe in umgekehrter Richtung.

5.3 Analyse nichtvektorieller Daten

Unter dem Begriff „nichtvektorielle Daten“ werde zahlreiche unterschiedliche Datentypen zusammengefasst. Diese werden jeweils gesondert betrachtet und mit speziellen Verfahren behandelt.

Hier werden wir uns nun das im vorangegangenen Kapitel vorgestellte Verfahren zur generellen und allgemeinen Behandlung nichtvektorieller Daten in einem Vektorraum genauer ansehen und auf seine Tauglichkeit hin überprüfen.

Im Gegensatz zu den üblichen, umfangreichen Verfahren wird die Behandlung nichtvektorieller Daten hier auf die Auswahl eines geeigneten Kernels reduziert. Das

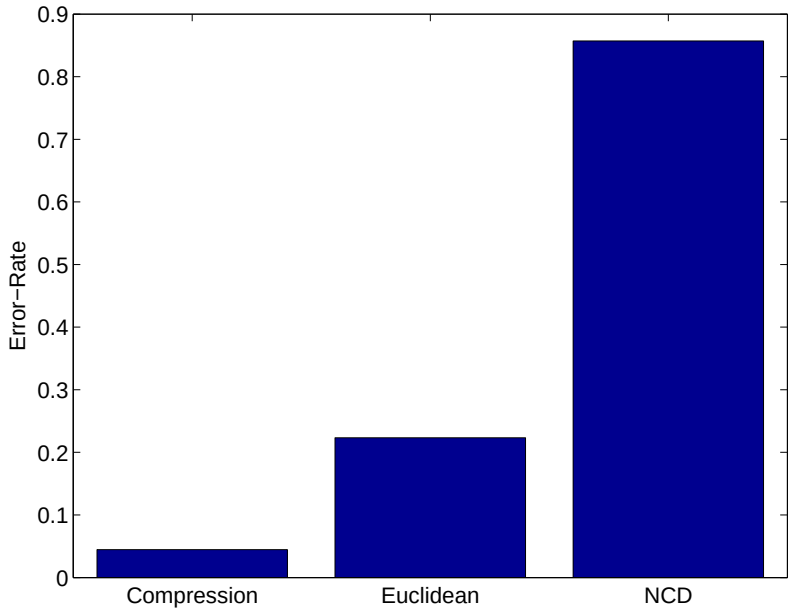


Abbildung 5.10: Fehlerraten für die Texturklassifikation der unterschiedlichen Methoden

ermöglicht es uns beliebige Daten – für die solch ein Kernel existiert – identisch, mit einer einzigen Methode zu behandeln. Um dies zu verdeutlichen, werden wir drei verschiedenen Datensätzen mit Hilfe einer Selbstorganisierenden Karte (engl. Self Organizing Map, SOM – Kohonen 2001) clustern. Eine SOM ist ein künstliches Neuronales Netz, das sich in großem Umfang der von Vektorräumen zur Verfüngung gestellten Methoden bedient. Es sollte noch erwähnt werden, dass unsere Wahl SOMs als Beispielmethode heranzuziehen als beliebig einzustufen ist: jedes andere Verfahren, das auf den Vektorraumaxiomen basiert, kann mit den hier vorgestellten Methoden behandelt werden. SOMs jedoch sind eine, vom Konzept her, sehr einfache Methode, die sehr weit verbreitet (Kaski et al. 1998) ist. Sie sind daher beinahe ideal geeignet, um als Beispiel zu fungieren.

5.3.1 Selbstorganisierende Karten

Selbstorganisierende Karten (SOM) dienen der Darstellung hochdimensionaler Datenverteilungen mit Hilfe niedrigdimensionaler Strukturen. Eine SOM besteht daher immer aus zwei Teilen, einer Menge an Gewichtsvektoren w_i, die Elementen des Datenraums entsprechen und einer korrespondierenden Menge an Knoten n_i, die niedrigdimensionale Struktur repräsentieren. Eine SOM ist daher ein aus untereinander verbundenen Knoten n_i bestehendes Netzwerk, das zu jedem Knoten einen Gewichtsvektor w_i bereihält. Das Ziel des SOM Algorithmus ist es, die Gewichtsvektoren so

anzupassen, dass das Netzwerk der Verteilung der Daten entspricht. Dies geschieht mit der SOM-Update Formel:

$$\Delta w_i = \alpha h_\sigma(n_i, n_j) \cdot (x - w_i).$$

Für jedes Datenelement x wird jeder Gewichtsvektor w_i in die Richtung dieser Datenelemente gedrückt, wobei die Intensität dieser Operation von der Entfernung des entsprechenden Knoten n_i zu der des Best-Matching Knotens n_j abhängt. Der Best-Matching Knoten ist der Knoten, der dem Datenelement am ähnlichsten ist. Je weiter ein Knoten vom Best-Matching Knoten entfernt ist, um so schwächer fällt die Update-Operation aus. Dies erhält die räumliche Nähe der Knoten im Netzwerk. Die Funktion

$$h_\sigma(n_i, n_j) = \exp\left(-\frac{\|n_i - n_j\|^2}{\sigma^2}\right),$$

üblicherweise eine unimodale Funktion, ist für die Berechnung der Intensität in Abhängigkeit von den beteiligten Knoten zuständig. Sie wird noch um die Schrittlänge α ergänzt.

Alle Berechnungen, die wir im Weiteren durchführen werden, entsprechen dem „Plain-Vanilla“ SOM Algorithmus, also exakt dem in Kohonen Buch vorgestellten. Die einzige Ausnahme hierzu ist die Norm $\|v\| = \langle v, v\rangle_{\mathcal{R}}^{1/2}$. Anstatt der üblichen ℓ^2 Norm verwenden wir die RDS-Norm. Solch eine „Anpassung“ der Norm ist von Kohonen explizit vorgesehen und entspricht daher der Idee der SOM.

5.3.2 Reuters Newswire Artikel

Unser erstes Beispiel bezieht sich auf Textdaten. Wir werden daher den Reuters-21578 Lewis (1987) Testdatensatz, eine Sammlung aus $21,578$ Reuters Newswire Texten, die von Experten in 120 Topics gegliedert wurden, verwenden. In diesem Experiment werden wir die Artikel mit einer SOM gruppieren und schauen wie die Gruppierungen der SOM mit denen der Experten übereinstimmen. Wir verwenden hierfür die Mod-Apte Aufteilung der Artikel wie in der Dokumentation zu dem Datensatz beschrieben (Lewis 1987). Diese Aufteilung besteht aus $9,603$ Trainingdokumenten, die wir als Datenbasis heranziehen werden. Wir wählen aus dieser Datenbasis zufällig 40 Wörterbuchelemente aus. Diese Zahl erhalten wir durch die Bestimmung des zu erwartenden Approximationsfehlers zu einer gegebenen Anzahl an Wörterbuchelementen (vgl. Abbildung 5.11). Die verbleibenden $9,563$ Datenelemente projezieren wir (wie in Abschnitt 4.5 beschrieben) auf das Wörterbuch mit Hilfe eines String Kernel Lodhi et al. (2002). Die auf diese Weise erzeugten $9,563$ RDS Elemente nutzen wir, um eine 40 $2D$ SOM zu trainieren. Die Knoten der SOM werden zufällig mit Datenelementen initialisiert. Der Update Algorithmus durchläuft $10,000$ Iterationen. Die Schrittweite $\alpha = 0.1$ ist während des gesamten Durchlaufs fest, genauso wie der Abstandsfaktor $\sigma = 4$ der Gaußfunktion h. Nach dem Training bestimmen wir zu jedem Gewichtsvektor noch ein Topic, dieses wählen wir aus der Menge der Topics der Datenvektoren aus, die diesen Gewichtsvektor als Best-Matching Knoten haben.

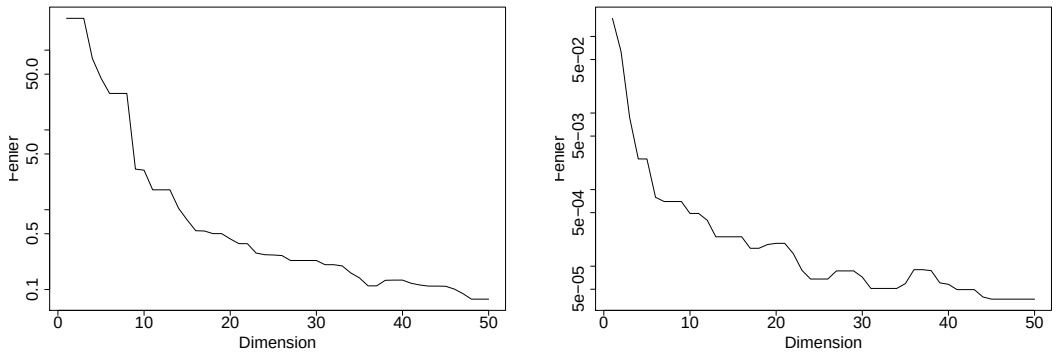


Abbildung 5.11: Der erwartete Feher der Projektion mit zunehmender Dimension, aufgetragen in Log-Skala. Auf der linken Seite sehen wir den Fehler des Farbhistogramm-Datensatzes. Auf der rechten den des Aktienkurs-Datensatzes. Die passende Anzahl an Wörterbuchelementen (oder Dimmensionen) lässt sich dort ablesen, wo die Kurve einen „Ellenbogen"(engl. elbow) hat. Dies ist ein häufig angwendetes Verfahren im Bereich der Dimensionreduzierung. Methoden wie PCA, MDS oder IsoMap Tenenbaum et al. (2000) bedienen sich dieses Verfahrens, um eine passende Anzahl von Eigenvektoren oder Projektionsdimensionen zu bestimmen.

Zu jedem Dokument in dem Datensatz bestimmen wir den ähnlichsten Gewichtsvektor und prüfen, inwiefern die Eingruppierung des Gewichtsvektors mit der des Dokumentes übereinstimmt. Die Ergebnisse sehen wir in Abbildung 5.12. Zu jedem Knoten und dessen Topic-Zuweisung bestimmen wir noch Precision, Recall und F_1, Größen, die im Information Retrieval große Bedeutung haben (Manning et al. 2008). Die Precision ($0.8925, 0.6310, 0.6529$), Recall ($0.8966, 0.7553, 0.5675$) und F_1 (0.8946, $0.6876, 0.6072$) Werte für die drei häufigsten Topics „earn", „acq" and „money-fx" deuten darauf hin, dass die meisten Dokumente mit dem Topic „earn" auch tatsächlich Knoten mit dem entsprechenden Topic zugewiesen werden. Des weiteren zeigen sie, dass Knoten die das Topic „earn" haben auch zum großen Teil nur für Datenvektoren mit diesem Topic als Best-Matching Knoden dienen. Bei den anderen Topics ist der Überlapp zwar größer, jedoch sollte bei einem Vergleich mit entsprechenden Ergebnissen, wie zum Beispiel denen von Joachims (1998), bedacht werden, dass es sich bei dem hier vorgestellten Verfahren um ein allgemeines Verfahren ohne jegliche datenspezifischen Anpassungen handelt.

5.3.3 Finanzmarkt Daten

In unserem zweiten Beispiel schauen wir uns Aktienkurse an. Obwohl dieser Datentyp sehr stark an vektorielle Daten erinnert, handelt es sich dabei um sogenannte Zeitreihen, die nur in seltenen Fällen mit Vektoren gleichgesetzt werden dürfen. Zeitreihen zeichnen sich in erster Linie durch den zeitlichen Ablauf von Werten aus, Ereignisse treten daher nicht immer zeitgleich auf. Bei Aktien unterschiedlicher Firmen ist

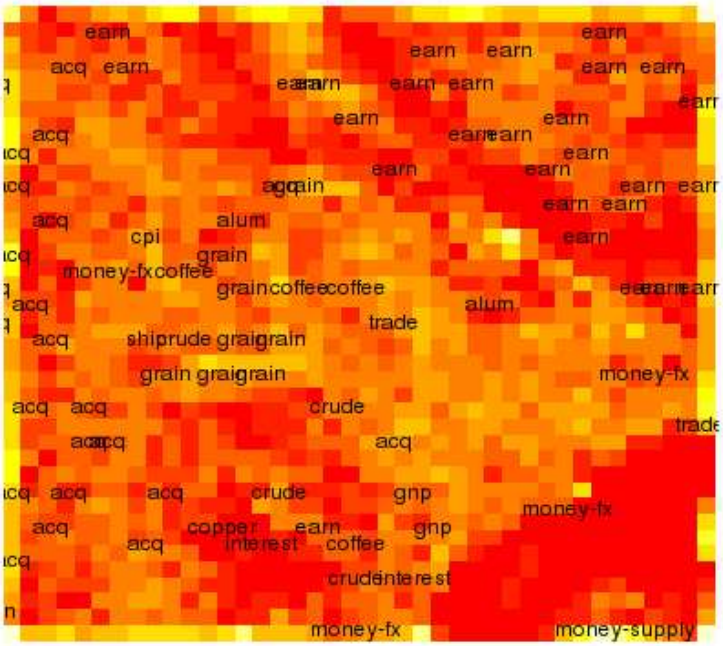


Abbildung 5.12: Die HeatMap Darstellung der Textdaten SOM. Die SOM wurde auf den $9,603$ Dokumenten des ModApte-Splits der Reuters-21578 Test Collection trainiert. Die Farben spiegeln die Dichte an den jeweiligen Knoten wieder, je heller die Farben, umso größer die Dichte. Die Bezeichnungen ergeben sich aus den dominierende Topics der jeweiligen Knoten.

es zum Beispiel sehr unwahrscheinlich, dass ein Aktiensplit zum gleichen Zeitpunkt stattfindet, Zyklen sind brachenabhängig und allgemeine Trends basieren auf der Wirtschaftslage einzelner Regionen. All das führt zu einem hohen Bedarf an Normalisierung und Registrierung (Ramsay und Silverman 2005). Dies bedeutet aber auch, dass die Zeitreihen intensiv vorverarbeitet werden müssen, was durch die Wahl eines geeigneten Kernel umgangen werden kann. Mit seiner Hilfe lassen sich Zeitreihen wie vektorielle Daten behandeln.

Als Datenset dienen uns die Kurse der 30 Aktien des Deutschen Aktienindex DAX aus der Zeit vom 01.01.2010 bis zum 01.08.2011. Die Daten wurden über die Yahoo Finance API heruntergeladen[5]. Um die Anzahl der Datenpunkte zu erhöhen, untertei-

[5]Yahoo Finance `http://finance.yahoo.com/` ist eine Webseite die programmatischen Zugriff auf Finanzdaten anbietet. Eine Dokumentation zu diesem Web Service findet man unter folgendem Link: `http://code.google.com/p/yahoo-finance-managed/wiki/YahooFinanceAPIs`.

len wir die Kursdaten in 50 Werte lange Sequenzen, was die Anzahl der Datenelemente auf 237 erhöht. Aus dieser Menge wählen wir 15 Wörterbuchelemente aus, auf die wir die übrigen 222 Datenelement projezieren. Als Kernel verwenden wir einen auf Dynamic Time Wrap (DTW – Sakoe und Chiba 1978) basierenden Kernel, der die zeitnormalisierte Distanz zwischen zwei Zeitreihen bestimmt. Auf diese Distanzen wenden wir dann eine Gaußfunktion an, um Ähnlichkeiten zu erhalten. Wie bei den Reuters Newswire Artikeln verwenden wir eine SOM mit einem $2D$ Netzwerk. Die SOM hat 15×15 Knoten und wird mit den Parametern $\alpha = 0.5$ und $\sigma = \sqrt{2}$ trainiert. Der Algorithmus ist identisch zu dem der Textdaten. Das Ergebnis der Berechungen sehen wir in Abbildung 5.13.

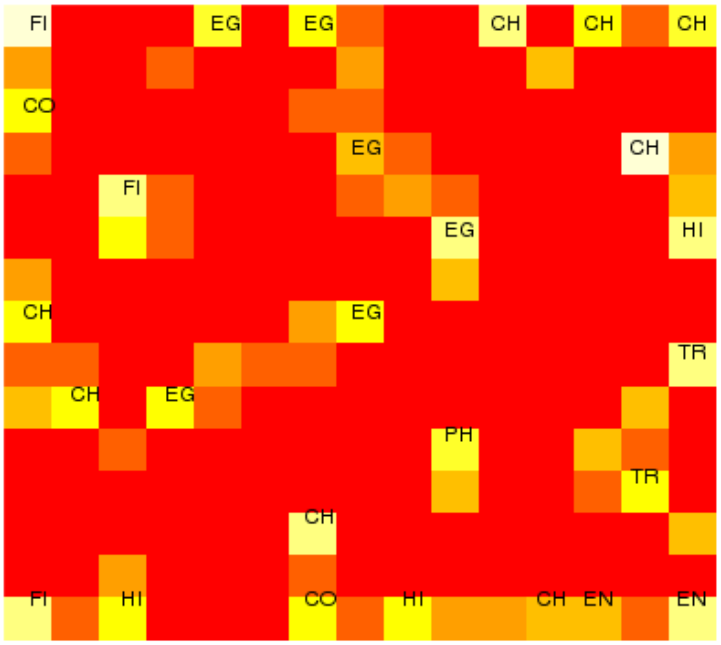


Abbildung 5.13: Die HeatMap Darstellung der $2D$ SOM, die auf den DAX Aktienkursen trainiert wurde. Die Label spiegeln die acht Branchen des Indexes (EnGineering, FInance, TRansportation and Communication, CHemical, PHarmaceutical, MEdical, COmmerce, und HIgh Tec) wieder. Die SOM zeigt deutlich die Gruppen der Hauptindustriezweige, wie zum Beipsiel Transportation, Chemistry und Engineering. Die anderen Gruppen haben stärkere Überlappungen, was hauptsächlich an Firmen wie Siemens oder Daimler liegt, die in mehreren Branchen tätig sind.

5.3.4 Bilddaten

In unserem letzten Beispiel gruppieren wir Farbbildhistogramme. Farbhistogramme sind vektorielle Daten, das heißt, wir könnten auch den SOM Algorithmus für vektorielle Daten verwenden, wie es zum Beispiel Möhrmann et al. (2011) gemacht haben. Wir werden jedoch in diesem Beispiel demonstrieren, dass der hier vorgeschlagene Algorithmus dieselben Ergebnisse liefert wie wir sie bei der Behandlung vektorieller Daten erwarten würden. Als Datensatz dient uns eine Untermenge der COIL Bilddatenbank (Nayar und Murase 1996). Die verwendeten Bilder zeigen vier unterschiedliche Objekte in unterschiedlichen Positionen (zu sehen in Abbildung 5.15). Von jedem dieser Bilder berechnen wir das Farbhistogramm (ein Vektor bestehend aus 36 Werten, jeder davon entspricht 10 Grad im HSV Farbkegel).

Die Datenbasis besteht aus 360 Histogrammen (72 für jedes der 5 Objekte). Wir wählen zufällig 17 Wörterbuchelemente aus dieser Datenbasis aus und verwenden das ursprüngliche Skalarprodukt, um die Daten auf den RDS zu projekzieren. Als SOM Parameter verwenden wir diesmal ein 10×10 $2D$ Network, $\alpha = 0.01$ und $\sigma = \sqrt{2}$. Der Algorithmus ist wieder identisch mit dem der beiden anderen Beispiele. Das Ergebnis sehen wir in Abbildung 5.14. Dort sehen wir auch klar die diskreten Gruppen für die einzelnen Objekte. Des weitere werden jedem Knoten nur Histogramme eines Objekts zugewiesen. Die Gruppierung ist also perfekt und somit identisch mit der Gruppierung von Möhrmann et al..

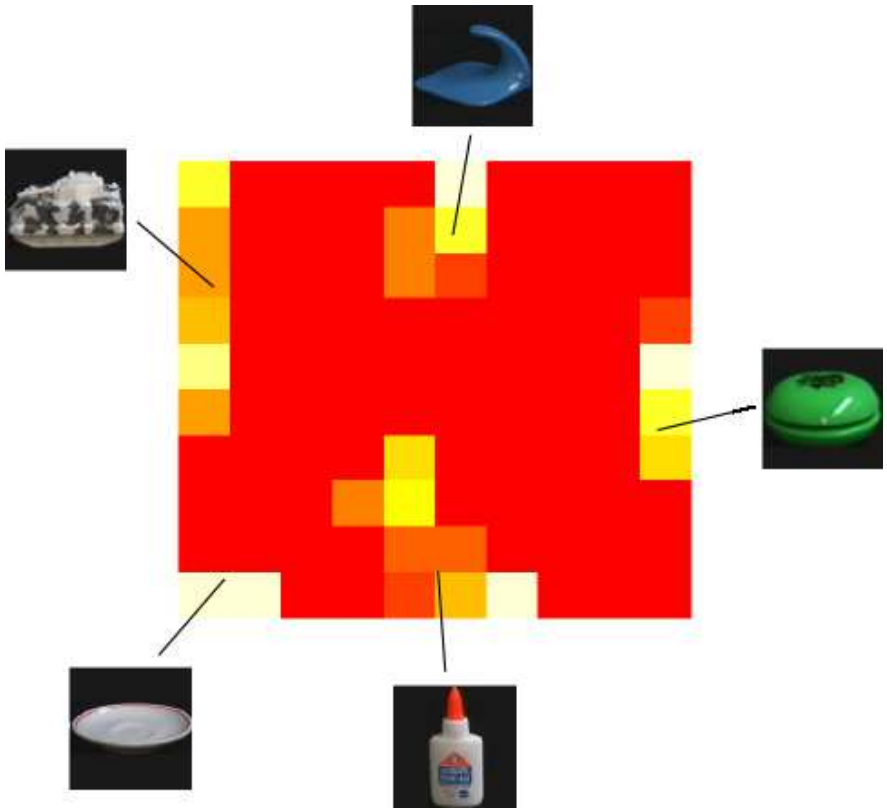


Abbildung 5.14: Die HeatMap Darstellung der $2D$ Bilddaten SOM mit 10×10 Knoten. Sie wurde auf den Farbhistogrammen der 320 Bilder der COIL Bilddatenbank trainiert. Jedes Rechteck steht für einen Knoten und dessen Farbe spiegelt die Dichte, also die Anzahl der Elemente, die dem korrespondierenden Gewichtsvektor zugeordnet wurden, wider. Je heller ein Knoten ist, um so mehr Histogramme weder diesem Knoten zugeordnet, dunklere Knoten repräsentieren weniger Histogramme.

Abbildung 5.15: Ein Objekt der COIL Bilddatenbank in vier verschiedenen Winkeln. Von diesen Bildern berechnen wir die Farbhistogramme für das Bilddatenexperiment.

KAPITEL

6 ZUSAMMENFASSUNG

Kompressionsbasierte Mustererkennung verbindet viele aktuelle Fragestellungen der Mustererkennung mit denen der Signalrepräsentation. Dabei werden nicht nur neue Anwendungsbereiche erschlossen, sondern auch konkrete Fragestellungen im Bereich der kompressionsbasierten Mustererkennung beantwortet. Zu diesen zählt zum Beispiel die Frage nach den erkannten Merkmalen dieser Verfahren.

Gerade die große Popularität der NCD führte die Forschung auf dem Bereich der kompressionsbasierten Mustererkennung jedoch lange Zeit in die Irre. Die Kolmogorovkomplexität stellt zwar eine ausgezeichnete Basis dar, um Informationsgehalt und Zufälligkeit zu studieren, sie ist jedoch in keinster Weise geeignet als Fundament für praktische Anwendungen, wie zum Beispiel Mustererkennung zu dienen.

6.1 NCD und Kolmogorovkomplexität

Wie wir in dieser Arbeit gesehen haben, gibt es sehr konkrete Schwierigkeiten, sowohl bei der praktischen Anwendung der Kolmogorovkomplexität, als auch bei der Interpretation der Ergebnisse. Kolmogorov betrachtet mit seinem Komplexitätsmaß abstrakte Konzepte und Zusammenhänge. Für eine praktisch Anwendung gibt es jedoch unüberwindbare Hürden, wie zum Beispiel in Korollar 2.1 zu sehen war.

Die zwei Beispiele 2.2 und 2.3 verdeutlichen nochmal die Schwierigkeiten, auf die man stößt, wenn man die allgemeine Version der Kolmogorovkomplexität als Basis für praktische Betrachtungen heranzieht. Durch eine geeignete Wahl der allgemeinen Kolmogorovkomplexität, lässt sich einer bestimmten Zeichenkette jegliche Komplexität zuweisen.

Des Weiteren haben wir uns den bekannten Zusammenhang zwischen der Kolmogorovkomplexität und dem Halteproblem angesehen. Dadurch haben wir uns noch einmal mit der Nichtberechenbarkeit dieses Komplexitätsmaßes beschäftigt. Da jedoch viele Experimente zur kompressionsbasierten Mustererkennung die Kolmogorovkomplexität als Motivation haben, betrachteten wir auch die NID und NCD. Diese Betrachtungen dienten als Basis für unsere darauf folgenden Experimente mit der Kompres-

sionsbasierten Mustererkennung. Dabei wollten wir das der NCD zugrundeliegende Funktionsprinzip aufdecken ohne uns allzu sehr von der Kolmogorovkomplexität ablenken zu lassen.

6.2 Praktische Anwendung der NCD

Theoretische Schwierigkeiten mit den zugrundeliegenden Komplexitätsmaßen haben zahlreiche Wissenschaftler nicht davon abgehalten, die NCD praktisch anzuwenden und damit Erfolg zu haben. Wir haben uns eine Beispielanwendung der NCD und drei bekannte Experimente näher angesehen, um zu verstehen was genau den Erfolg dieser Methode ausmacht.

Bei der Beispielanwendung war das Ziel, Daten aus verschiedenen Quellen zusammenzuführen. Dies erfordert eine Zuordnung der einzelnen Datensätze, was sehr schwierig ist, wenn keine eindeutige Kennung oder fehlerhafte Daten präsent sind. Wie wir gesehen haben, bietet die NCD hier eine einfache und schnelle Methode zur Datenzusammenführung.

Es war die Aufgabe der Beispielanwendung, die NCD in Aktion zu sehen. Ziel der Experimente war es, tiefe Einblicke in die Funktionsweise der NCD zu erhalten. Dabei wurde deutlich, dass es sich bei den gefundenen Mustern um eher kurze Zeichenketten, wie zum Beispiel Worte handelte. Die damit codierte Information reichte aus, um Zusammenhänge, wie zum Beispiel die Autorenschaft von Texten oder die Verwandschaft von Säugetieren, zu erkennen.

In einem weiteren Experiment untersuchten wir die Anwendung dieser Arbeitsweise auch auf Bildern und fanden heraus, dass die erkannten Muster nur selten ausreichten, um semantische Zusammenhänge zu detektieren. Nun stellte sich die Frage, ob diese Unzulänglichkeiten systemimmanent sind, oder nur auf eine unzureichende Anwendung zurückzuführen sind.

6.3 Kompressionsbasierte Mustererkennung

Im Kapitel Kompressionsbasierte Mustererkennung haben wir Kompression und Mustererkennung so formalisiert, dass es uns möglich war, Ergebnisse auch ohne die Kolmogorovkomplexität zu erklären. Dabei lag der Fokus, im Gegensatz zur Motivation der NCD, nicht auf einer unberechenbaren Maschine, sondern auf zwei Methoden, einer zur Bestimmung von Mustern (der Kompression) und einer zur Berechnung von Ähnlichkeiten. Das Ergebnis davon ist eine Theorie zur kompressionsbasierten Mustererkennung, die alle bestehenden Ergebnisse erklären (und reproduzieren) kann und auch Raum für neue Entwicklungen lässt.

Wir haben uns in diesem Kontext mit Räumen über redundanten Wörterbüchern beschäftigt. Dabei haben wir gesehen, wie man auch bezüglich redundanter „Basissysteme“ rechnen kann. Diese bildeten die Grundlage für die kompressionsbasierte

Mustererkennung, da dort ebenfalls hochgradig redundante Wörterbücher erzeugt und damit Daten codiert werden. In drei Beispielen haben wir uns diese Analogie konkret angesehen. Die NCD ließ sich auch mit dieser Theorie erklären, was uns gleich zu einer Reihe von Experimenten führte, in der die NCD als sehr spezifischer Vertreter kompressionsbasierter Verfahren mit anderen Methoden dieser Gattung verglichen wurde.

Bei unseren Experimenten zur kompressionsbasierten Mustererkennung lag der Fokus diesmal eindeutig auf kontinuierlichen Daten. Diese stellen für die NCD immer noch eine Schwierigkeit dar, und wir wollten sehen, ob allgemeine kompressionsbasierte Verfahren hier Vorteile haben. Wir haben daher mit Zeitreihendaten angefangen und diese mit Hilfe von SVMs mit kompressionsbasiertem Kernel klassifiziert. Der dabei verwendete Phoneme Datensatz hat einige Herausforderungen für die Repräsentationsmethode zu bieten. Die Ergebnisse demonstrieren die Leistungsfähigkeit kompressionsbasierter Methoden und auch deren vielseitige Anwendbarkeit (in diesem Fall als SVM Kernel). Die darauf folgenden Experimente wendeten kompressionsbasierte Verfahren auf Texte (zum Vergleich mit der NCD) und auf Bilder (als anspruchsvollem kontinuierlichen Anwendungsfall) an. Dabei konnten die kompressionsbasierten Verfahren in allen Bereichen überzeugen. Neben den positiven Ergebnissen konnten wir jedoch auch Einblicke in die Arbeitsweise der Verfahren gewinnen. So zeigte besonders das Experimnet auf den Texturdaten, wie wichtig die Wahl eines geeigneten Wörterbuch ist und wieviel Einfluss diese Wahl auf das Ergebnis hat.

6.4 Ausblick

Kompressionsbasierte Musterkennung stellt eine Kombination aus Methoden der Funktionsrepräsentation und Musterkennung dar. Gerade die Forschung im Bereich des Sparse Codings ist im Augenblick sehr aktiv, sodass hier in den nächsten Jahren noch sehr viele Neuerungen zu erwarten sind. Kompressionsbasierte Mustererkennung erlaubt es uns, direkt von diesen Erkenntnissen zu profitieren.

Ein Aspekt, der in dieser Arbeit noch nicht berücksichtigt wurde, ist das Erlernen oder die Generierung von Wörterbüchern. Da die Wörterbücher jedoch von so zentraler Bedeutung für die Qualität der Erkennung sind, kann man noch auf einige spannende Erkenntnisse hoffen.

Die, in vorangegangenen Kapiteln vorgestellten Räume über redundanten Wörterbüchern haben neben der Mustererkennung auch noch weiteres Potential, dessen Behandlung in dieser Arbeit leider keinen Platz fand. Auch hier kann man gespannt sein, welche weiteren Ergebnisse die Forschung noch zu Tage bringen wird.

LITERATURVERZEICHNIS

Amitay et al. 2007

AMITAY, Einat ; YOGEV, Sivan ; YOM-TOV, Elad: Serial Sharers: Detecting Split Identities of Web Authors. In: STEIN, Benno (Hrsg.) ; KOPPEL, Moshe (Hrsg.) ; STAMATATOS, Efstathios (Hrsg.): *PAN* Bd. 276, CEUR-WS.org, 2007 (CEUR Workshop Proceedings) 38

Baeza-Yates und Ribeiro-Neto 1999

BAEZA-YATES, Ricardo ; RIBEIRO-NETO, Berthier: *Modern Information Retrieval*. Addison Wesley, 1999. – ISBN 020139829X 41

Benedetto et al. 2002

BENEDETTO, Dario ; CAGLIOTI, Emanuele ; LORETO, Vittorio: Language Trees and Zipping. In: *Physical Review Letters* 88 (2002), Nr. 4 47, 48, 50

Bennett et al. 1998

BENNETT, C.H. ; GACS, P. ; LI, Ming ; VITÀNYI, Paul M. ; ZUREK, W.H.: Information distance. In: *Information Theory, IEEE Transactions on* 44 (1998), Jul, Nr. 4, S. 1407–1423. – ISSN 0018–9448 22, 23, 24, 26, 27

Bilenko und Mooney 2003

BILENKO, Mikhail ; MOONEY, Raymond J.: Adaptive duplicate detection using learnable string similarity measures. In: *KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2003, S. 39–48 38

Bishop 2006

BISHOP, Christopher M.: *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 2006. – ISBN 0387310738 102, 103

Brodatz 1966

BRODATZ, Phil: *Textures: A Photographic Album for Artists and Designers*. New York : Dover Publications Inc., 1966 109

Burkovski 2009

BURKOVSKI, Andre: *Analyse und Visualisierung von NCD-Features*. Stuttgart, Deutschland, Universität Stuttgart, Diplomarbeit, 2009 27, 33, 34, 65

Burkovski et al. 2009

BURKOVSKI, Andre ; KLENK, Sebastian ; HEIDEMANN, Gunther: *The Length Delimiting Dictionary Distance: an abstract compression-based Distance Measure*. 2009. – Unpublished 47, 48, 54, 64

Candès und Romberg 2006
CANDÈS, Emmanuel J. ; ROMBERG, Justin: Sparsity and incoherence in compressive sampling. In: *Inverse Problems* 23 (2006), S. 969–985 72

Candès et al. 2006
CANDÈS, Emmanuel J. ; ROMBERG, Justin ; TAO, Terence: Stable signal recovery from incomplete and inaccurate measurements. In: *Communications on Pure and Applied Mathematics* 59 (2006), Nr. 8, S. 1207–1223 72

Cao et al. 1998
CAO, Ying ; JANKE, Axel ; WADDELL, Peter J. ; WESTERMAN, Michael ; TAKENAKA, Osamu ; MURATA, Shigenori ; OKADA, Norihiro ; PÄBO, Svante ; HASEGAWA, Masami: Conflict Among Individual Mitochondrial Proteins in Resolving the Phylogeny of Eutherian Orders. In: *Journal of Molecular Evolution* 47 (1998), September, Nr. 3, S. 307–322 98

Cebrian et al. 2007
CEBRIAN, M. ; ALFONSECA, M. ; ORTEGA, A.: The Normalized Compression Distance Is Resistant to Noise. In: *Information Theory, IEEE Transactions on* 53 (2007), May, Nr. 5, S. 1895–1900 38

Chakrabarti 2009
CHAKRABARTI, Soumen (Hrsg.): *Data mining*. Amsterdam : Elsevier, Morgan Kaufmann, 2009. – ISBN 9780123746290 10, 39

Chen et al. 1996
CHEN, Scott S. ; DONOHO, David L. ; SAUNDERS, Michael A.: Atomic Decomposition by Basis Pursuit. In: *SIAM Journal on Scientific Computing* 20 (1996), S. 33–61 74

Chen et al. 2000
CHEN, Xin ; KWONG, Sam ; LI, Ming: A compression algorithm for DNA sequences and its applications in genome comparison. In: *RECOMB '00: Proceedings of the fourth annual international conference on Computational molecular biology*. New York, NY, USA, 2000. – ISBN 1581131860, S. 107 98

Christen 2007
CHRISTEN, Peter: A two-step classification approach to unsupervised record linkage. In: *AusDM '07: Proceedings of the sixth Australasian conference on Data mining and analytics*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2007, S. 111–119 39

Christen und Goiser 2007
CHRISTEN, Peter ; GOISER, Karl: Quality and Complexity Measures for Data Linkage and Deduplication. In: GUILLET, Fabrice (Hrsg.) ; HAMILTON, Howard J. (Hrsg.): *Quality Measures in Data Mining* Bd. 43. Springer, 2007, S. 127–151 40

Cilibrasi und Vitànyi 2005
CILIBRASI, Rudi ; VITÀNYI, Paul M.: Clustering by compression. In: *Information Theory,*

IEEE Transactions on 51 (2005), Nr. 4 3, 4, 6, 10, 13, 22, 24, 25, 26, 27, 29, 38, 54, 88, 92, 93, 98, 100, 101

Conway 1990
CONWAY, J.B.: *A Course in Functional Analysis*. Springer-Verlag, 1990 (Graduate texts in mathematics). – ISBN 9780387972459 76

Cormen et al. 2001
CORMEN, Thomas H. ; STEIN, Clifford ; RIVEST, Ronald L. ; LEISERSON, Charles E.: *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001. – ISBN 0070131511 31

Cover und Thomas 2006
COVER, Thomas M. ; THOMAS, Joy A.: *Elements of information theory*. 2. Auflage. Hoboken, NJ : Wiley-Interscience, 2006 (A Wiley-Interscience publication). – ISBN 0471241954 18, 21, 22, 25, 32

Daubechies 2004
DAUBECHIES, Ingrid: *Ten lectures on wavelets*. 8. print. Philadelphia, Pa. : Society for Industrial and Applied Mathematics, 2004 (Regional conference series in applied mathematics ; 61). – ISBN 0898712742 91

Davis et al. 1997
DAVIS, Geoff ; MALLAT, Stèphane G. ; AVELLANEDA, Marco: Adaptive greedy approximations. In: *Journal of Constructive Approximation* 13 (1997), Dec, Nr. 1, S. 57–98 71

Donoho 2006
DONOHO, David L.: For most large underdetermined systems of equations, the minimal l1-norm near-solution approximates the sparsest near-solution. In: *Communications on Pure and Applied Mathematics* 59 (2006), Dec, Nr. 7, S. 907–934 72

Elmagarmid et al. 2007
ELMAGARMID, Ahmed K. ; IPEIROTIS, Panagiotis G. ; VERYKIOS, Vassilios S.: Duplicate Record Detection: A Survey. In: *Knowledge and Data Engineering, IEEE Transactions on* 19 (2007), Jan, Nr. 1, S. 1–16 38, 39

Ethnologue
http://www.ethnologue.com 47

Fellegi und Sunter 1969
FELLEGI, Ivan P. ; SUNTER, Alan B.: A Theory for Record Linkage. In: *Journal of the American Statistical Association* 64 (1969), Nr. 328, S. 1183–1210 38

Ferraty und Vieu 2006
FERRATY, Frédéric ; VIEU, Philippe: *Nonparametric Functional Data Analysis: Theory and Practice*. Secaucus, NJ, USA : Springer New York, Inc., 2006 (Springer Series in Statistics). – ISBN 0387303693 102, 104

Fitch und Margoliash 1967
FITCH, Walter M. ; MARGOLIASH, Emanuel: Construction of phylogenetic trees. In: *Science* 155 (1967), Nr. 3760, S. 279–284 48

Fritz et al. 2010
FRITZ, Peter ; KLENK, Sebastian ; GOLETZ, Sven ; GERTEIS, Andreas ; SIMON, Wolfgang ; BRINKMANN, Friedhelm ; HEIDEMANN, Else ; LÜTTGEN, Elisabeth ; OTT, German ; ALSCHER, Mark D. ; SCHWAB, Matthias ; DIPPON, Jürgen: Clinical Impacts of Histological Sub-Typing Primary Breast Cancer. In: *Anticancer Research* 30 (2010), Nr. 12 II

Fritz et al. 2008
FRITZ, Peter ; KLENK, Sebastian ; W SCHROTH, L A. S Amaral A. S Amaral ; GERTEIS, Andreas ; SIMON, Wolfgang ; BRINKMANN, Friedhelm ; HEIDEMANN, Else ; LÜTTGEN, Elisabeth ; OTT, German ; ALSCHER, Mark D. ; SCHWAB, Matthias ; BRAUCH, Hiltrud: Langzeit Follow-up Datenbanken als unverzichtbares Instrumentarium molekularbiologischer Untersuchungen beim Mammakarzinom. In: *Tagungsband der 53. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e. V. (GMDS)*. Stuttgart, Germany, 2008 II

Gärtner 2003
GÄRTNER, Thomas: A survey of kernels for structured data. In: *SIGKDD Explor. Newsl.* 5 (2003), July, S. 49–58 85

Goiser und Christen 2006
GOISER, Karl ; CHRISTEN, Peter: Towards automated record linkage. In: *AusDM '06: Proceedings of the fifth Australasian conference on Data mining and analystics*. Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2006, S. 23–31 39

Gribonval und Nielsen 2007
GRIBONVAL, Rèmi ; NIELSEN, Morten: Highly sparse representations from dictionaries are unique and independent of the sparseness measure. In: *Applied and Computational Harmonic Analysis* 22 (2007), Nr. 3, S. 335 – 355 76

Grünwald 2004
GRÜNWALD, Peter: A tutorial introduction to the minimum description length principle. In: *CoRR* math.ST/0406077 (2004) 24

Han und Kamber 2001
HAN, Jiawei ; KAMBER, Micheline: *Data mining*. Morgan Kaufmann Publ., 2001. – ISBN 1558604898 39

Hastie et al. 2002
HASTIE, Trevor J. ; TIBSHIRANI, Robert J. ; FRIEDMAN, Jerome H.: *The elements of statistical learning*. Corrected print. Springer, 2002. – ISBN 0387952845 102, 104, 106, 108, 111

Haupt und Nowak 2006
HAUPT, Jarvis ; NOWAK, Robert: Signal reconstruction from noisy random projections. In: *Information Theory, IEEE Transactions on* 52 (2006), S. 4036–4048 72

Heidemann und Klenk 2007
HEIDEMANN, Gunther ; KLENK, Sebastian: Visual Analytics for Image Retrieval. In: MIKUT, R. (Hrsg.) ; REISCHL, M. (Hrsg.): *Proc. 17th Workshop Computational Intelligence*, 2007, S. 15–24 II

Heidemann und Ritter 2008
HEIDEMANN, Gunther ; RITTER, Helge: On the Contribution of Compression to Visual Pattern Recognition. In: *Proc. 3rd Int'l Conf. on Comp. Vision Theory and Applications* Bd. 2. Funchal, Madeira - Portugal, 2008, S. 83–89 29, 38

Hernández und Stolfo 1998
HERNÁNDEZ, Mauricio A. ; STOLFO, Salvatore J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem. In: *Data Mining and Knowledge Discovery* 2 (1998), Nr. 1, S. 9–37 38

Huffman 1952
HUFFMAN, David A.: A Method for the Construction of Minimum-Redundancy Codes. In: *Proceedings of the Institute of Radio Engineers* 40 (1952), Sep, Nr. 9, S. 1098–1101 30

Husmeier et al. 2004
HUSMEIER, Dirk ; DYBOWSKI, Richard ; ROBERTS, Stephen: *Probabilistic Modelling in Bioinformatics and Medical Informatics*. Springer, 2004. – ISBN 1852337788 98

Imo et al. 2008a
IMO, Johannes ; KLENK, Sebastian ; HEIDEMANN, Gunther: Interactive Feature Visualization for Image Retrieval. In: *Proc. 19th International Conference on Pattern Recognition (ICPR08)*. Tampa, USA, 2008 II

Imo et al. 2008b
IMO, Johannes ; KLENK, Sebastian ; HEIDEMANN, Gunther: Visualization of Color and Texture Features. In: *Proc. 1st International Workshop on Super Visualization (IWSV08)*. Kos, Greece, 2008 II

Joachims 1998
JOACHIMS, Thorsten: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *Eur. Conf. Mach. Learn. (ECML)*. Berlin : Springer, 1998, S. 137–142 115

Kaski et al. 1998
KASKI, Samuel ; KANGAS, Jari ; KOHONEN, Teuvo: Bibliography of Self-Organizing Map (SOM) Papers: 1981–1997. In: *Neural Comput. Surv.* 1 (1998), S. 102–350 113

Keogh et al. 2004
KEOGH, Eamonn ; LONARDI, Stefano ; RATANAMAHATANA, Chotirat A.: Towards parameter-free data mining. In: *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2004. – ISBN 1581138881, S. 206–215 10, 24, 26, 27

Kimball und Caserta 2004
KIMBALL, Ralph ; CASERTA, Joe: *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleanin*. John Wiley & Sons, 2004. – ISBN 0764567578 38, 39

Kleene 1971
KLEENE, Stephen C.: *Introduction to metamathematics*. 6. Auflage. Groningen : Wolters-Noordhoff, 1971 (Bibliotheca mathematica ; 1). – ISBN 0720421039 15, 20

Klenk et al. 2008
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: Interaktive Überlebenszeitanalyse: Datenbankdesign und -implementierung. In: *Tagungsband der 53. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie e. V. (GMDS)*. Stuttgart, Germany, 2008 II

Klenk et al. 2009a
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: Interactive survival analysis with the OCDM system: From development to application. In: *Information Systems Frontiers* 11 (2009), Nr. 4, S. 391–403 II

Klenk et al. 2010a
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: Determining Patient Similarity in Medical Social Networks. In: *Proc. 1st International Workshop on Web Science and Information Exchange in the Medical Web (MedEx 2010)*. Raleigh, USA, 2010 II

Klenk et al. 2010b
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: Relevance Based Visualization of Large Cancer Patient Populations. In: *Proc. 1st ACM International Health Informatics Symposium (IHI 2010)*. Washington, USA, 2010 II

Klenk et al. 2011a
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: Medizinisches Data Mining mit dem OCDM System. In: *Tagungsband der DVMD Jahrestagung 2011*. Hannover, Germany, 2011 II

Klenk et al. 2011b
KLENK, Sebastian ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: A Personalized Medical Information System. In: *Proc. 3rd International Workshop on Software Engineering in Health Care*. Hawaii, USA, 2011 II

Klenk und Heidemann 2008
KLENK, Sebastian ; HEIDEMANN, Gunther: A new method for Principal Component

Analysis of high-dimensional data using Compressive Sensing. In: *Proc. 4th International Conference on Data Mining (DMIN'08)*. Las Vegas, USA, 2008 68

Klenk und Heidemann 2009
KLENK, Sebastian ; HEIDEMANN, Gunther: A Sparse coding based similarity measure. In: *Proc. 5th International Conference on Data Mining (DMIN'09)*. Las Vegas, USA, 2009 68

Klenk et al. 2011c
KLENK, Sebastian ; MÖHRMANN, Julia ; BURKOVSKI, Andre ; DIPPON, Jürgen ; FRITZ, Peter ; HEIDEMANN, Gunther: A Personalized Health Information System to foster Preventive Medicine. In: *Proc. 12th International Conference on Bioinformatics & Computational Biology (BIOCOMP'11)*. Las Vegas, USA, 2011 II

Klenk et al. 2009b
KLENK, Sebastian ; THOM, Dennis ; HEIDEMANN, Gunther: The Normalized Compression Distance as a Distance Measure in Entity Identification. In: PERNER, Petra (Hrsg.): *Advances in Data Mining: 9th Industrial Conference on Data Mining ICDM2009*, Springer, 2009 29, 36

Kohonen 2001
KOHONEN, Teuvo: *Self-organizing maps*. 3. Auflage. Berlin : Springer, 2001 (Springer series in information sciences ; 30). – ISBN 3540679219 113, 114

Kolmogorov 1993
KOLMOGOROV, Andrey N. ; SIRJAEV, Albert N. (Hrsg.): *Mathematics and its applications : Soviet series ; 27*. Bd. 3: *Selected works of A. N. Kolmogorov*. Dordrecht : Kluwer, 1993. – ISBN 9027727988 15, 16

Lempel und Ziv 1976
LEMPEL, Abraham ; ZIV, Jacob: On the Complexity of Finite Sequences. In: *Information Theory, IEEE Transactions on* 22 (1976), Jan, Nr. 1, S. 75 – 81. – ISSN 0018–9448 14, 33, 34, 90

Lewis 1987
LEWIS, David D.: *Reuters-21578, Distribution 1.0 test collection*. 1987. – The Reuters-21578, Distribution 1.0 test collection is available from `http://www.daviddlewis.com/resources/testcollections/reuters21578`. 79, 114

Li et al. 2001
LI, Ming ; BADGER, Jonathan H. ; CHEN, Xin ; KWONG, Sam ; KEARNEY, Paul ; ZHANG, Haoyong: An information-based sequence distance and its application to whole mitochondrial genome phylogeny . In: *Bioinformatics* 17 (2001), Nr. 2, S. 149–154 98

Li et al. 2004
LI, Ming ; CHEN, Xin ; LI, Xin ; MA, Bin ; VITÀNYI, Paul M.: The similarity metric. In: *Information Theory, IEEE Transactions on* 50 (2004), Dec, Nr. 12, S. 3250–3264 6, 13, 15, 23, 24, 26, 29, 98

Li und Vitànyi 1993
LI, Ming ; VITÀNYI, Paul M.: *An introduction to Kolmogorov complexity and its applications*. New York : Springer, 1993 (Texts and monographs in computer science). – ISBN 0387940537 14, 22, 24, 27

Lodhi et al. 2002
LODHI, Huma ; SAUNDERS, Craig ; SHAWE-TAYLOR, John ; CRISTIANINI, Nello ; WATKINS, Chris: Text classification using string kernels. In: *Journal of Machine Learning Research* 2 (2002), March, S. 419–444. – ISSN 1532–4435 85, 114

Luenberger 1969
LUENBERGER, David G.: *Optimization by vector space methods*. New York : Wiley, 1969 (Series in decision and control) 75

Lüneburg 2002
LÜNEBURG, Heinz: *Rekursive Funktionen*. Berlin : Springer, 2002 (Springer-Lehrbuch). – ISBN 3540430946 15

Lynch 1974
LYNCH, Nancy: Approximations to the halting problem. In: *Journal of Computer and System Sciences* 9 (1974), Nr. 2, S. 143 – 150 24

Mallat 2009
MALLAT, Stèphane G.: *A wavelet tour of signal processing : the sparse way*. 3. Auflage. Amsterdam : Academic Press Elsevier, 2009. – ISBN 0123743702 8, 72, 73, 74

Mallat und Zhang 1993
MALLAT, Stèphane G. ; ZHANG, Zhifeng: Matching pursuits with time-frequency dictionaries. In: *Signal Processing, IEEE Transactions on* 41 (1993), Dec, Nr. 12, S. 3397–3415 71, 72, 73

Manning et al. 2008
MANNING, Christopher D. ; RAGHAVAN, Prabhakar ; SCHÜTZE, Hinrich: *Introduction to Information Retrieval*. 1. Cambridge University Press, 2008. – ISBN 0521865719 115

Matlab 2008
Documentation for MATLAB R2008a. `http://www.mathworks.com/access/helpdesk/help/techdoc/index.html`. Version: 2008. – The MathWorks Inc. 109

McCallum et al. 2000
MCCALLUM, Andrew ; NIGAM, Kamal ; UNGAR, Lyle H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: *KDD '00: Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2000, S. 169–178 40

Meyer 1992
MEYER, Yves: *Wavelets and operators*. Cambridge : Cambridge Univ. Pr., 1992 (Cambridge studies in advanced mathematics ; 37). – ISBN 0521420008 111

Möhrmann et al. 2011
MÖHRMANN, Julia ; BERNSTEIN, Stefan ; SCHLEGEL, Thomas ; WERNER, Günter ; HEIDEMANN, Gunther: Improving the Usability of Hierarchical Representations for Interactively Labeling Large Image Data Sets. In: *Lecture Notes in Computer Science* 6761 (2011), S. 618–627 118

Navarro 2001
NAVARRO, Gonzalo: A guided tour to approximate string matching. In: *ACM Comput. Surv.* 33 (2001), Nr. 1, S. 31–88 38

Nayar und Murase 1996
NAYAR ; MURASE, H.: Columbia Object Image Library: COIL-100 / Department of Computer Science, Columbia University. 1996 (CUCS-006-96). – Forschungsbericht 64, 118

Nocedal und Wright 2000
NOCEDAL, Jorge ; WRIGHT, Stephen J.: *Numerical optimization*. Corr. print. New York : Springer, 2000 (Springer series in operations research). – ISBN 0387987932 75

Papadopoulos 2005
PAPADOPOULOS, Athanase: *Metric spaces, convexity and nonpositive curvature*. Zürich : European Mathematical Society, 2005 (IRMA lectures in mathematics and theoretical physics ; 6). – ISBN 3037190108 76

R Development Core Team 2010
R DEVELOPMENT CORE TEAM: *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing, 2010. `http://www.R-project.org`. – ISBN 3900051070 48

Ramsay und Silverman 2005
RAMSAY, James O. ; SILVERMAN, Bernard W.: *Functional Data Analysis*. Berlin : Springer, 2005 116

Riesz und Sz.-Nagy 1990
RIESZ, Frigyes ; SZ.-NAGY, Béla: *Functional analysis*. New York : Dover Publications Inc., 1990 (Dover Books on Advanced Mathematics). – xii+504 S. – ISBN 0486662896. – Translated from the second French edition by Leo F. Boron, Reprint of the 1955 original 85, 93

Ripley 2007
RIPLEY, Brian D.: *Pattern recognition and neural networks*. 7. print. Cambridge Univ. Press, 2007. – ISBN 0521717700 102, 107

Rogers 1967
ROGERS, Hartley: *Theory of recursive functions and effective computability*. New York : McGraw-Hill, 1967 (McGraw-Hill series in higher mathematics) 15, 16, 17, 18, 20, 21

Rudin 1991
RUDIN, Walter: *Functional analysis*. 2. Auflage. Boston, Mass. : McGraw-Hill, 1991 (International series in pure and applied mathematics) 76, 81, 88

Runkler 2010
RUNKLER, Thomas A.: *Data Mining : Methoden und Algorithmen intelligenter Datenanalyse*. Wiesbaden : Vieweg+Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden, 2010 (SpringerLink). – ISBN 9783834893536 39

Sakoe und Chiba 1978
SAKOE, Hiroaki ; CHIBA, Seibi: Dynamic programming algorithm optimization for spoken word recognition. In: *Signal Processing, IEEE Transactions on* 26 (1978), feb, Nr. 1, S. 43 – 49 80, 95, 117

Sarawagi und Bhamidipaty 2002
SARAWAGI, Sunita ; BHAMIDIPATY, Anuradha: Interactive deduplication using active learning. In: *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. New York, NY, USA : ACM, 2002, S. 269–278 38

Sayood 2000
SAYOOD, Khalid: *Introduction to data compression*. 2. Auflage. San Francisco, Calif. [u.a.] : Morgan Kaufmann, 2000. – ISBN 1558605584 14

Schmidt 1907
SCHMIDT, Erhard: Zur Theorie der linearen und nichtlinearen Integralgleichungen. I. Teil: Entwicklung willkürlicher Funktionen nach Systemen vorgeschriebener. In: *Mathematische Annalen* 63 (1907), Nr. 5, S. 433–476. – `http://www.digizeitschriften.de/dms/img/?PPN=PPN235181684_0063&DMDID=dmdlog45` 84, 85

Schnorr 1974
SCHNORR, Claus-Peter: Optimal Enumerations and Optimal Gödel Numberings. In: *Theory of Computing Systems* 8 (1974), S. 182–191 18

Schölkopf und Smola 2002
SCHÖLKOPF, Bernhard ; SMOLA, Alexander J.: *Learning with Kernels*. Cambridge, Mass. [u.a.] : MIT Press, 2002 (Adaptive computation and machine learning series). – ISBN 0262194759 70

Sculley und Brodley 2006
SCULLEY, David ; BRODLEY, Carla E.: Compression and Machine Learning: A New Perspective on Feature Space Vectors. In: *DCC '06: Proceedings of the Data Compression Conference*. Washington, DC, USA : IEEE Computer Society, 2006. – ISBN 0769525458, S. 332–332 11, 89

Shannon 1948
SHANNON, Claude E.: A Mathematical Theory of Communication. In: *The Bell System Technical Journal* 27 (1948), S. 379–423,623–656 13

Shoukry et al. 2010
SHOUKRY, Laila ; KLENK, Sebastian ; HEIDEMANN, Gunther: MPEG-7 Feature Visualization for CBIR Systems. In: *Proc. International Conference on Computer Theory and Applications (ICCTA'2010)*. Alexandria, Egypt, 2010 II

Sigurd et al. 2004
SIGURD, Bengt ; EEG-OLOFSSON, Mats ; VAN WEIJER, Joost: Word length, sentence length and frequency Zipf revisited. In: *Studia Linguistica* 58 (2004), Nr. 1, S. 37–52 62

Simovici et al. 2008
SIMOVICI, Dan A. ; DJERABA, Chabane ; JAIN, Lakhmi C. ; WU, Xindong: *Mathematical Tools for Data Mining : Set Theory, Partial Orders, Combinatorics*. London : Springer-Verlag London Limited, 2008 (Advanced Information and Knowledge Processing). – ISBN 9781848002012 69

Tenenbaum et al. 2000
TENENBAUM, Josh B. ; DE SILVA, Vin ; LANGFORD, John C.: A global geometric framework for nonlinear dimensionality reduction. In: *Science* 290 (2000), Dez, S. 2319–2323 82, 115

Thom 2008
THOM, Dennis: *Visualisierung von Mustern Kompressionsbasierter Ähnlichkeitsmetriken*. Stuttgart, Deutschland, Universität Stuttgart, Studienarbeit, 2008 26, 27, 64, 65

Thom 2010
THOM, Dennis: *Kompressionsbasierte Mustererkennung auf Bildern*. Stuttgart, Deutschland, Universität Stuttgart, Diplomarbeit, 2010 26

Tichonov und Arsenin 1977
TICHONOV, Andrej N. ; ARSENIN, Vasilij J.: *Solutions of ill-posed problems*. Washington, D.C. : Winston, 1977 (Scripta series in mathematics A Halsted Press book). – ISBN 0470991240. – Translated from Russ. 11, 70, 71

Turk und Pentland 1991
TURK, Matthew ; PENTLAND, Alex: Eigenfaces for recognition. In: *J. Cognitive Neuroscience* 3 (1991), January, S. 71–86 79

Vapnik 1998
VAPNIK, Vladimir N.: *Statistical learning theory*. New York : Wiley, 1998 (Adaptive and learning systems for signal processing, communications, and control). – ISBN 0–471–03003–1 11, 70, 102

Vert et al. 2004
In: VERT, Jean-Philippe ; SAIGO, Hiroto ; AKUTSU, Tatsuya: *Local Alignment Kernels for Biological Sequences*. Cambridge : MIT Press, 2004, S. 131–153 85

Vishwanathan et al. 2010
VISHWANATHAN, Vishy ; SCHRAUDOLPH, Nicol N. ; KONDOR, Risi I. ; BORGWARDT,

Karsten M.: Graph Kernels. In: *Journal of Machine Learning Research* 11 (2010), S. 1201–1242 85

Welch 1984
WELCH, Terry A.: A Technique for High-Performance Data Compression. In: *Computer* 17 (1984), Jun, Nr. 6, S. 8 –19. – ISSN 0018–9162 34, 35

Whitehead und Russel 1957
WHITEHEAD, Alfred N. ; RUSSEL, Bertrand: *Principia mathematica*. Bd. 1:. 2. Auflage. Cambridge : Univ. Press, 1957 20

Wikipedia 2010a
WIKIPEDIA: *Bzip2 — Wikipedia,*. `http://en.wikipedia.org/w/index.php?title=Bzip2&oldid=379434299`. Version: 2010. – [Online; Stand 18. August 2010] 30

Wikipedia 2010b
WIKIPEDIA: *Einwegfunktion. — Wikipedia,*. `http://de.wikipedia.org/w/index.php?title=Einwegfunktion&oldid=75524194`. Version: 2010. – [Online; Stand 26. Juli 2010] 23

Wikipedia 2010c
WIKIPEDIA: *Gzip — Wikipedia,*. `http://de.wikipedia.org/w/index.php?title=Gzip&oldid=77475290`. Version: 2010. – [Online; Stand 22. September 2010] 48

Wikipedia 2010d
WIKIPEDIA: *Theory — Wikipedia,*. `http://en.wikipedia.org/w/index.php?title=Theory&oldid=335780631`. Version: 2010. – [Online; Stand 4. Januar 2010] 67

Winkler 2006
WINKLER, William E.: Overview of record linkage and current research directions / US Bureau of the Census. 2006 (RRS2006/02). – Forschungsbericht 38

Yan et al. 2007
YAN, Su ; LEE, Dongwon ; KAN, Min-Yen ; GILES, Lee C.: Adaptive sorted neighborhood methods for efficient record linkage. In: *JCDL '07: Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*. New York, NY, USA : ACM, 2007, S. 185–194 39

Zhao 2007
ZHAO, Huimin: Semantic matching across heterogeneous data sources. In: *Commun. ACM* 50 (2007), Nr. 1, S. 45–50 38, 39

Zhao und Ram 2005
ZHAO, Huimin ; RAM, Sudha: Entity identification for heterogeneous database integration: a multiple classifier system approach and empirical evaluation. In: *Inf. Syst.* 30 (2005), Nr. 2, S. 119–132 38

Zhao und Ram 2008
ZHAO, Huimin ; RAM, Sudha: Entity matching across heterogeneous data sources: An approach based on constrained cascade generalization. In: *Data & Knowledge Engineering* (2008). – In Press, Corrected Proof, Available online 4 May 2008 38, 39

Ziv und Lempel 1977
ZIV, Jacob ; LEMPEL, Abraham: A universal algorithm for sequential data compression. In: *Information Theory, IEEE Transactions on* 23 (1977), May, Nr. 3, S. 337 – 343. – ISSN 0018–9448 10, 11, 29, 30, 34, 48

Ziv und Lempel 1978
ZIV, Jacob ; LEMPEL, Abraham: Compression of individual sequences via variable-rate coding. In: *Information Theory, IEEE Transactions on* 24 (1978), Sep, Nr. 5, S. 530 – 536. – ISSN 0018–9448 30, 34