Sabine Schulte im Walde, Helmut Schmid, Wiebke Wagner,
Christian Hying, and Christian Scheible:

A Clustering Approach to Automatic Verb Classification incorporating Selectional Preferences: Model, Implementation, and User Manual

Universität Stuttgart

SinSpeC

Working Papers of the SFB 732

"Incremental Specification in Context"

Universität Stuttgart

SinSpeC 07 (2010) ISSN 1867-3082

SinSpeC issues do not appear on a strict schedule.

Volume **07 (December 2010)**

Authors: Sabine Schulte im Walde, Helmut Schmid, Wiebke Wagner, Christian Hying, and Christian Scheible
Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart
Azenbergstraße 12
70174 Stuttgart
Germany

Series Editors: Artemis Alexiadou
Universität Stuttgart
Institut für Lingusitik/Anglistik
Heilbronner Strasse 7
D-70174 Stuttgart

Hinrich Schütze
Universität Stuttgart
Institut für maschinelle Sprachverarbeitung
Azenbergstrasse 12
D-70174 Stuttgart

Published by **O**nline **P**ublikationsverbund der **U**niversität **S**tuttgart (OPUS)

Published 2010

ISSN **1867-3082**

About *SinSpeC*

SinSpeC are the Working Papers of the Sonderforschungsbereich (SFB) 732 "Incremental Specification in Context". The SFB 732 is a collaboratory research center at the University of Stuttgart and has been funded by the German Research Foundation (DFG) since July 1, 2006.

The SFB 732 brings together scientists from the areas of linguistics, computational linguistics and signal processing at the University of Stuttgart. Their common scientific goals are to achieve a better understanding of the mechanisms that lead to ambiguity control/disambiguation as well as the enrichment of missing/incomplete information and to develop methods that are able to fully describe these mechanisms.

For further information about the SFB please visit:

*http://*www.uni-stuttgart.de/linguistik/sfb732/

SinSpeC aims at publishing ongoing work within the SFB in a fast and uncomplicated way in order to make the results of our work here known to the scientific community and strengthen our international relationships. It publishes papers by the staff of the SFB as well as papers by visiting scholars or invited scholars.

SinSpeC is available online on the above website.

Contact Information:

Director of the SFB 732:

Prof. Dr. Artemis Alexiadou
artemis@ifla.uni-stuttgart.de

Coordinator of the SFB 732:

Dr. Sabine Mohr
sabine@ifla.uni-stuttgart.de

SFB 732
Universität Stuttgart
Keplerstraße 17
D-70174 Stuttgart

Phone: 0711/685-83115
Fax: 0711/685-83120

Contents

Chapter 1

Introduction

In recent years, the computational linguistics community has developed an impressive number of semantic verb classifications, i.e., classifications that generalise over verbs according to their semantic properties. Intuitive examples of such classifications are the MOTION WITH A VEHICLE class, including verbs such as *drive, fly, row*, etc., or the BREAK A SOLID SURFACE WITH AN INSTRUMENT class, including verbs such as *break, crush, fracture, smash*, etc. Semantic verb classifications are of great interest to computational linguistics, specifically regarding the pervasive problem of data sparseness in the processing of natural language. Up to now, such classifications have been used in applications such as word sense disambiguation (Dorr and Jones, 1996; Kohomban and Lee, 2005), machine translation (Prescher et al., 2000; Koehn and Hoang, 2007), document classification (Klavans and Kan, 1998), and in statistical lexical acquisition in general (Rooth et al., 1999; Merlo and Stevenson, 2001; Korhonen, 2002; Schulte im Walde, 2006).

Given that the creation of semantic verb classifications is not an end task in itself, but depends on the application scenario of the classification, we find various approaches to an automatic induction of semantic verb classifications. For example, Siegel and McKeown (2000) used several machine learning algorithms to perform an automatic aspectual classification of English verbs into event and stative verbs. Merlo and Stevenson (2001) presented an automatic classification of three types of English intransitive verbs, based on argument structure and heuristics to thematic relations. Pereira et al. (1993) and Rooth et al. (1999) relied on the Expectation-Maximisation algorithm to induce soft clusters of verbs, based on the verbs' direct object nouns. Similarly, Korhonen et al. (2003) relied on the Information Bottleneck (Tishby et al., 1999) and subcategorisation frame types to induce soft verb clusters.

This report presents two variations of an innovative, complex approach to semantic verb classes that relies on selectional preferences as verb properties. The underlying linguistic assumption for this verb class model is that verbs which agree on their selectional preferences belong to a common semantic class. The model is implemented as a soft-clustering approach, in order to capture the polysemy of the verbs. The training procedure uses the Expectation-Maximisation (EM)

algorithm (Baum, 1972) to iteratively improve the probabilistic parameters of the model, and applies the Minimum Description Length (MDL) principle (Rissanen, 1978) to induce WordNet-based selectional preferences for arguments within subcategorisation frames. One variation of the MDL principle replicates a standard MDL approach by Li and Abe (1998), the other variation presents an improved pruning strategy that outperforms the standard implementation considerably. Our model is potentially useful for lexical induction (e.g., verb senses, subcategorisation and selectional preferences, collocations, and verb alternations), and for NLP applications in sparse data situations. We demonstrate the usefulness of the model by a standard evaluation (pseudo-word disambiguation), and three applications (selectional preference induction, verb sense disambiguation, and semi-supervised sense labelling).

The report is structured as follows. We first introduce the two versions of our clustering model, accompanied by the evaluation and the applications (Chapter 2). The other main parts of the report cover details of the implementation (Chapter 3), and a user manual (Chapter 4).

Chapter 2

Clustering Model

2.1 Verb Class Model

2.1.1 Probabilistic Model

Our probabilistic model of verb classes groups verbs into clusters with similar subcategorisation frames and similar selectional preferences. Verbs may be assigned to several clusters (soft clustering) which allows the model to describe the subcategorisation properties of several verb readings separately. The number of clusters is defined in advance, but the assignment of the verbs to the clusters is learnt during training. It is assumed that all verb readings belonging to one cluster have similar subcategorisation and selectional properties. The selectional preferences are expressed in terms of semantic concepts from WordNet, rather than a set of individual words. Finally, the model assumes that the different arguments are mutually independent for all subcategorisation frames of a cluster. From the last assumption, it follows that any statistical dependency between the arguments of a verb has to be explained by multiple readings.

The statistical model is characterised by the following equation which defines the probability of a verb v with a subcategorisation frame f and arguments $a_1, ..., a_{n_f}$:

$$p(v, f, a_1, ..., a_{n_f}) = \sum_c p(c)\ p(v|c)\ p(f|c)\ * \prod_{i=1}^{n_f} \sum_{r \in R} p(r|c, f, i)\ p(a_i|r)$$

The model describes a stochastic process which generates a verb-argument tuple like ⟨*speak, subj-pp.to, professor, audience*⟩ by

1. selecting some cluster c, e.g. c_3 (which might correspond to a set of *communication* verbs), with probability $p(c_3)$,

2. selecting a verb v, here the verb *speak*, from cluster c_3 with probability $p(\text{speak}|c_3)$,

3. selecting a subcategorisation frame f, here *subj-pp.to*, with probability $p(\text{subj-pp.to}|c_3)$, (Note that the frame probability only depends on the cluster, and not on the verb.)

4. selecting a WordNet concept r for each argument slot, e.g. *person* for the first slot with probability $p(\text{person}|c_3, \text{subj-pp.to}, 1)$ and *social group* for the second slot with probability $p(\text{social group}|c_3, \text{subj-pp.to}, 2)$,

5. selecting a word a_i to instantiate each concept as argument i; in our example, we might choose *professor* for *person* with probability $p(\text{professor}|\text{person})$ and *audience* for *social group* with probability $p(\text{audience}|\text{social group})$.

The model contains two *hidden variables*, namely the clusters c and the selectional preferences r. In order to obtain the overall probability of a given verb-argument tuple, we have to sum over all possible values of these hidden variables.

The assumption that the arguments are independent of the verb given the cluster is essential for obtaining a clustering algorithm because it forces the EM algorithm to make the verbs within a cluster as similar as possible.[1] The assumption that the different arguments of a verb are mutually independent is important to reduce the parameter set to a tractable size.

The fact that verbs select for concepts rather than individual words also reduces the number of parameters and helps to avoid sparse data problems. In order to find the right level of specifity for the concepts used as selectional preferences, we apply the Minimum Description Length (MDL) principle (Rissanen, 1978) , cf. Section 2.1.3.

The probabilities $p(r|c, f, i)$ and $p(a|r)$ mentioned above are not directly estimated. Instead, we follow the approach by Abney and Light (1999) and turn WordNet into a Hidden Markov model (HMM). We create a new pseudo-concept for each WordNet noun and add it as a hyponym to each synset containing this word. In addition, we assign a probability to each hypernymy–hyponymy transition, such that the probabilities of the hyponymy links of a synset sum up to 1. The pseudo-concept nodes emit the respective word with a probability of 1, whereas the regular concept nodes are non-emitting nodes. The probability of a path in such a WordNet HMM is the product of the probabilities of the transitions within the path. The probability $p(a|r)$ is then defined as the sum of the probabilities of all paths from the concept r to the word a in the "back-off" model. There is only one back-off model for these slot-independent argument head probabilities $p(a|r)$. Similarly, we create a separate partial WordNet HMM for each argument slot $\langle c, f, i\rangle$ which encodes the selectional preferences of that slot. The HMM contains only nodes for the WordNet concepts that the slot selects for, and the dominating concepts. The probability $p(r|c, f, i)$ is the total probability of all paths from the top-most WordNet concept *entity* to the terminal node r.

[1] The EM algorithm adjusts the model parameters in such a way that the probability assigned to the training tuples is maximised. Given the model constraints, the data probability can only be maximised by making the verbs within a cluster as similar to each other as possible, regarding their required arguments.

By combining slot-specific probability models $p(r|c, f, i)$ for selectional preferences which are expressed as WordNet concepts with a general probability model $p(a|r)$ for the realisation of concepts by nouns, we can exactly model the selectional preferences of each slot and – at the same time – generalise over the individual words observed in the training data.

2.1.2 EM Training

The clustering model is trained on verb-argument tuples of the form described above, i.e., consisting of a verb and a subcategorisation frame, plus the nominal[2] heads of the arguments. The tuples may be extracted from parsed data, or from a treebank. Because of the hidden variables, the model is trained iteratively with the Expectation-Maximisation algorithm (Baum, 1972). The parameters are randomly initialised and then re-estimated with the Inside-Outside algorithm (Lari and Young, 1990) which is an instance of the EM algorithm for training Probabilistic Context-Free Grammars (PCFGs).

The PCFG training algorithm is applicable here because we can define a PCFG for each of our models which generates the same verb-argument tuples with the same probability. The PCFG is defined as follows:

(1) The start symbol is TOP.

(2) For each cluster c, we add a rule TOP $\rightarrow$ V_c A_c whose probability is $p(c)$.

(3) For each verb v in cluster c, we add a rule $\mathrm{V}_c \rightarrow$ v with probability $p(v|c)$.

(4) For each subcategorisation frame f of cluster c with length n, we add a rule $\mathrm{A}_c \rightarrow$ f $\mathrm{R}_{c,f,1,entity}$ ... $\mathrm{R}_{c,f,n,entity}$ with probability $p(f|c)$.

(5) For each transition from a node r to a node r' in the selectional preference model for slot i of the subcategorisation frame f of cluster c, we add a rule $\mathrm{R}_{c,f,i,r} \rightarrow \mathrm{R}_{c,f,i,r'}$ whose probability is the transition probability from r to r' in the respective WordNet-HMM.

(6) For each terminal node r in the selectional preference model, we add a rule $\mathrm{R}_{c,f,i,r} \rightarrow \mathrm{R}_r$ whose probability is 1. With this rule, we "jump" from the selectional preference model to the corresponding node in the a priori model.

(7) For each transition from a node r to a node r' in the a priori model, we add a rule $\mathrm{R}_r \rightarrow \mathrm{R}_{r'}$ whose probability is the transition probability from r to r' in the a priori WordNet-HMM.

(8) For each word node a in the a priori model, we add a rule $\mathrm{R}_a \rightarrow$ a whose probability is 1.

Based on the above definitions, a partial "parse" for ⟨*speak subj-pp.to professor audience*⟩, referring to cluster 3 and one possible WordNet path, is shown in Figure 2.1. The connections

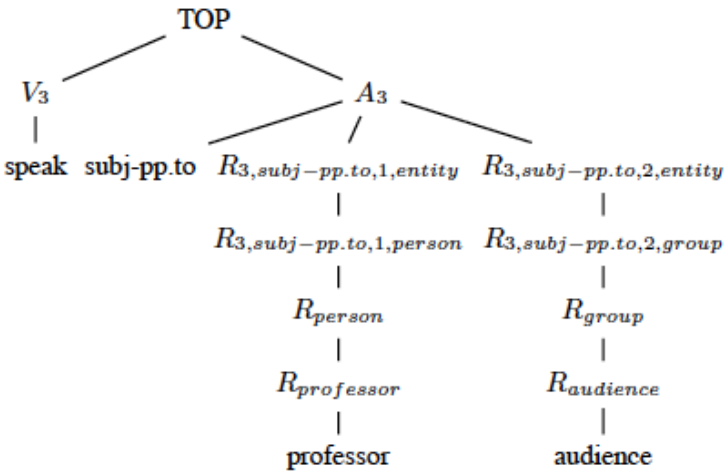

Figure 2.1: Example parse tree.

within R_3 ($R_{3,...,entity}$–$R_{3,...,person/group}$) and within R ($R_{person/group}$–$R_{professor/audience}$) refer to sequential applications of rule types (5) and (7), respectively.

The EM training algorithm maximises the likelihood of the training data.

2.1.3 MDL Principle

A model with a large number of fine-grained concepts as selectional preferences assigns a higher likelihood to the training data than a model with a small number of more coarse-grained concepts, because the larger number of parameters will better model idiosyncracies of the training data. Consequently, the EM algorithm will prefer very fine-grained concepts and – due to sparse data problems – will overfit the training data. In order to find selectional preferences with optimal granularity, we apply the Minimum Description Length principle, an approach from Information Theory. According to the MDL principle, the model with minimal *description length* should be chosen. The description length itself is the sum of the *model length* and the *data length*. The model length is the number of bits needed to encode the model and its parameters, and the data length is the number of bits required to encode the training data with the given model. According to coding theory, an optimal encoding uses $-log_2 p$ bits, on average, to encode data whose probability is p. Usually, the model length increases and the data length decreases as more parameters are added to a model. The MDL principle finds a compromise between the size of the model and the accuracy of the data description. Our selectional preference model incorporates two variants of the MDL principle to determine selectional preferences of verbs and their arguments, by means of a concept hierarchy ordered by hypernym/hyponym relations. The

[2]In principle, any word classes that are covered by WordNet could be included, but the taxonomy for the nouns is the most elaborated, so arguments with lexical heads other than nouns (e.g., subcategorised clauses) are not included in the selectional preference induction.

two variants are described below. Both models use WordNet 3.0 as the concept hierarchy, and comprise one (complete) back-off WordNet model for the lexical head probabilities $p(a|r)$ and one (partial) model for each selectional probability distribution $p(r|c, f, i)$, cf. Section 2.1.1.

MDL Model 1: ***Node Pruning Strategy (NPS)***

Our first selectional preference model relies on a standard approach by Li and Abe (1998). Given a set of nouns within a specific argument slot as a sample, the approach finds the cut in a concept hierarchy which minimises the sum of encoding both the model and the data. A *cut* is defined as a set of concepts (nodes) in the concept hierarchy that defines a partition of the "leaf" concepts (the lowest concepts in the hierarchy), viewing each concept in the cut as representing the set of all leaf concepts it dominates. Because the cut represents a partition of the hierarchy, a concept node that is decided to be expanded/pruned, can only be expanded/pruned by all of its hyponyms. This is the core difference to Model 2 (see below), where all hyponyms of a concept node are considered individually when they are expanded or pruned.

The *model length (ML)* is defined as

$$ML = \frac{k}{2} * log_2 |S|,$$

with k the number of concepts in the partial hierarchy between the top concept and the concepts in the cut, and $|S|$ the sample size, i.e., the total frequency of the data set. The *data length (DL)* is defined as

$$DL = -\sum_{n \in S} log_2\, p(n).$$

The probability of a noun $p(n)$ is determined by dividing the total probability of the concept class the noun belongs to, $p(concept)$, by the size of that class, $|concept|$, i.e., the number of nouns that are dominated by that concept:

$$p(n) = \frac{p(concept)}{|concept|}.$$

The higher the concept within the hierarchy, the more nouns receive an equal probability, and the greater is the data length.

The probability of the concept class in turn is determined by dividing the frequency of the concept class $f(concept)$ by the sample size:

$$p(concept) = \frac{f(concept)}{|S|},$$

where $f(concept)$ is calculated by upward propagation of the frequencies of the nominal lexemes from the data sample through the hierarchy. For example, if the nouns *coffee, tea, milk* appeared

with frequencies $25, 50, 3$ within a specific argument slot, then their hypernym concept *beverage* would be assigned a frequency of 78, and these 78 would be propagated further upwards to the next hypernyms, etc. As a result, each concept class is assigned a fraction of the frequency of the whole data set (and the top concept receives the total frequency of the data set). For calculating $p(concept)$ and the overall data length, though, only the concept classes within the cut through the hierarchy are relevant.

MDL Model 2: ***Edge Pruning Strategy (EPS)***

The edge pruning strategy (EPS) differs from the node pruning strategy (NPS) in three ways:

- NPS either prunes all edges of a concept node or none, whereas EPS allows the different edges of a concept node to its hyponyms to be pruned individually.

  Take the GermaNet concept $\langle Tier \rangle$ (animal) as an example, which has 16 hyponyms, among them $\langle Futtertier \rangle$ (food animal) and $\langle Wildtier \rangle$ (wild animal). With NPS, we cannot have a transition from $\langle Tier \rangle$ to $\langle Wildtier \rangle$ in the selectional preference model without also having transitions to all the other hyponyms. With EPS, we can.

- NPS makes the simplifying assumption that all nouns covered by some selectional preference concept have the same probability (i.e. $p(dog|\langle animal \rangle) = p(Weimaraner|\langle animal \rangle)$). EPS instead assumes (in accordance with the clustering model) that the nouns are distributed according to the back-off distribution $p(a|r)$. This allows EPS to use more general concepts as selectional preferences than NPS when the concept contains both frequent and infrequent nouns. NPS overestimates in this case the data description length and prunes too cautiously.

- NPS assumes that all transition probabilities are stored with the same precision (i.e. number of bits), which only depends on the size of the training corpus. EPS assumes that the precision of the transition probabilities depends on the frequency of the respective hypernym concept, i.e. the probabilities of transitions from frequent concepts are assumed to be stored with a higher precision than the probabilities of transitions from infrequent nodes.[3] This means that EPS prunes transitions from infrequent nodes less aggressively than NPS.

In the following, we describe how the model description length and the data description length change when an additional transition (edge) from a hypernym to a hyponym is added.

Adding a new transition k from hypernym s_k to hyponym t_k increases the model length by:

$$\Delta_k^M = 1 + \log_2 M + 0.5 \log_2 N.$$

[3] Parameter estimates for edges originating from frequent nodes are more reliable and therefore it makes sense to use a higher precision.

$\log_2 M$ bits (where M is the number of hyponyms of s_k) are needed to identify the hyponym to which the transition leads[4]. $0.5 \log_2 N$ bits (where N is the frequency of s_k) are used to store the probability of the transition, and 1 bit is required to indicate that the (new) target node is a terminal node.[5]

The new transition changes the probability of the transition $s_k \rightarrow t_k$. When the transition k is missing in the selectional preference model, we use the probability p_k^A of this transition in the back-off model instead. The back-off probability is scaled by the back-off factor α in order to ensure that the probabilities of all transition from s_k sum up to 1. After adding transition k to the selectional preference model, we use a relative frequency estimate $\frac{f_k}{N}$ instead of the back-off probability.

$$\begin{aligned} p_{old}(s_k \rightarrow t_k) &= \alpha_{old}\, p_k^A \\ p_{new}(s_k \rightarrow t_k) &= \frac{f_k}{N} \end{aligned}$$

f_k is the estimated frequency of the transition according to the E step of the EM algorithm.

Adding transition k changes the back-off factor α and therefore the probabilities of the pruned transitions as follows:

$$\begin{aligned} p_{old}(s_j \rightarrow t_j) &= \alpha_{old}\, p_j^A \\ p_{new}(s_j \rightarrow t_j) &= \alpha_{new}\, p_j^A \end{aligned}$$

Overall adding transition k decreases the data length by:

$$\begin{aligned} \Delta_k^D &= f_k(\log p_{new}(s_j \rightarrow t_j) - \log p_{old}(s_k \rightarrow t_k)) + \sum_{j \in E'_p} f_j(\log p_{new}(s_j \rightarrow t_j) - \log p_{old}(s_j \rightarrow t_j)) \\ &= f_k \log \frac{f_k}{N \alpha_{old}\, p_k^A} + \sum_{j \in E'_p} f_j \log \frac{\alpha_{new}}{\alpha_{old}} \end{aligned}$$

where $E_{p'}$ is the set of pruned transitions at s_k.

The total change in description length caused by adding transition k is the difference between the change in model description length and the change in data description length $\Delta_k^M - \Delta_k^D$. Since we use MDL for pruning, we remove a transition if the difference is positive.

2.1.4 Combining EM and MDL

The training procedure combines the EM training with the MDL principle. In the following, we summarise the interaction with respect to the two MDL models.

[4] We have to store a number in the range $1...M$ which requires $\lceil log_2 M \rceil$ bits. We ignore the ceiling operation in our formula for simplicity.

[5] This extra bit is not needed if the target node is already part of the selectional preference model. We ignore this for simplicity.

MDL Model 1: ***Node Pruning Strategy (NPS)***

1. The probabilities of a verb class model with c classes and a pre-defined set of verbs and frames are initialised randomly. The selectional preference models start out with the most general WordNet concept only, i.e., the partial WordNet hierarchies underlying the probabilities $p(r|c, f, i)$ initially only contain the concept r for *entity*.

2. The model is trained for a pre-defined number of iterations. In each iteration, not only the model probabilities are re-estimated and maximised (as done by EM), but also the cuts through the concept hierarchies that represent the various selectional preference models are re-assessed. In each iteration, the following steps are performed.

   (a) The partial WordNet hierarchies that represent the selectional preference models are expanded to include the hyponyms of the respective leaf concepts of the partial hierarchies. I.e., in the first iteration, all models are expanded towards the hyponyms of *entity*, and in subsequent iterations each selectional preference model is expanded to include the hyponyms of the leaf nodes in the partial hierarchies resulting from the previous iteration. This expansion step allows the selection models to become more and more detailed, as the training proceeds and the verb clusters (and their selectional preferences) become increasingly specific.

   (b) The training tuples are processed: For each tuple, a PCFG parse forest as indicated by Figure 2.1 is done, and the Inside-Outside algorithm is applied to estimate the frequencies of the "parse tree rules", given the current model probabilities.

   (c) The MDL principle is applied to each selectional preference model: Starting from the respective leaf concepts in the partial hierarchies, MDL is calculated to compare each set of hyponym concepts that share a hypernym with the respective hypernym concept. If the MDL is lower for the set of hyponyms than for the hypernym, the hyponyms are left in the partial hierarchy. Otherwise the expansion of the hypernym towards the hyponyms is undone and we continue recursively upwards the hierarchy, calculating MDL to compare the former hypernym and its co-hyponyms with the next upper hypernym, etc. The recursion allows the training algorithm to remove nodes which were added in earlier iterations and are no longer relevant. It stops if the MDL is lower for the hyponyms than for the hypernym.
   This step results in selectional preference models that minimally contain the top concept *entity*, and maximally contain the partial WordNet hierarchy between *entity* and the concept classes that have been expanded within this iteration.

   (d) The probabilities of the verb class model are maximised based on the frequency estimates obtained in step (b).

MDL Model 2: ***Edge Pruning Strategy (EPS)***

The model using the edge pruning strategy (EPS) is basically trained in the same way as the NPS model, with one important difference. The NPS model actually removes pruned transitions during training and uses the back-off probability for pruned transitions in the next iteration. This makes it difficult for the model to learn the selectional preferences because most of the information learned about the differences between the selectional preferences of two clusters is wiped out by the pruning at the end of each iteration.

There are two different EPS versions: **EPS-a** prunes in the same ways as NPS and replaces the probabilities of pruned links with the respective back-off probabilities during training. **EPS-b** preserves the probabilities of those pruned links which will be reinserted in the expansion step of the next training iteration. These are links originating from nodes with at least one unpruned incoming link.

2.1.5 Related Work

Our model is an extension of and thus most closely related to the latent semantic clustering (LSC) model (Rooth et al., 1999) for verb-argument pairs $\langle v, a \rangle$ which defines their probability as follows:

$$p(v, a) = \sum_c p(c)\ p(v|c)\ p(a|c)$$

In comparison to our model, the LSC model only considers a single argument (such as direct objects), or a fixed number of arguments from one particular subcategorisation frame, whereas our model defines a probability distribution over all subcategorisation frames. Furthermore, our model specifies selectional preferences in terms of general WordNet concepts rather than sets of individual words.

In a similar vein, our model is both similar and distinct in comparison to the soft clustering approaches by Pereira et al. (1993) and Korhonen et al. (2003). Pereira et al. (1993) suggested deterministic annealing to cluster verb-argument pairs into classes of verbs and nouns. On the one hand, their model is asymmetric, thus not giving the same interpretation power to verbs and arguments; on the other hand, the model provides a more fine-grained clustering for nouns, in the form of an additional hierarchical structure of the noun clusters. Korhonen et al. (2003) used verb-frame pairs (instead of verb-argument pairs) to cluster verbs relying on the Information Bottleneck (Tishby et al., 1999). They had a focus on the interpretation of verbal polysemy as represented by the soft clusters. The main difference of our model in comparison to the above two models is, again, that we incorporate selectional preferences (rather than individual words, or subcategorisation frames).

In addition to the above soft-clustering models, various approaches towards semantic verb classification have relied on hard-clustering models, thus simplifying the notion of verbal polysemy.

Two large-scale approaches of this kind are Schulte im Walde (2006), who used k-Means on verb subcategorisation frames and verbal arguments to cluster verbs semantically, and Joanis et al. (2008?), who applied Support Vector Machines to a variety of verb features, including subcategorisation slots, tense, voice, and an approximation to animacy. To the best of our knowledge, Schulte im Walde (2006) is the only hard-clustering approach that previously incorporated selectional preferences as verb features. However, her model was not soft-clustering, and she only used a simple approach to represent selectional preferences by WordNet's top-level concepts, instead of making use of the whole hierarchy and more sophisticated methods, as in the current paper.

Last but not least, there are other models of selectional preferences than the MDL model we used in our paper. With a recent exception (Erk, 2007), whose selectional preference model exploits similarity-based models, Most such models also rely on the WordNet hierarchy (Resnik, 1997; Abney and Light, 1999; Ciaramita and Johnson, 2000; Clark and Weir, 2002). Brockmann and Lapata (2003) compared some of the models against human judgements on the acceptability of sentences, and demonstrated that the models were significantly correlated with human ratings, and that no model performed best; rather, the different methods are suited for different argument relations.

2.2 Evaluation

The clustering model in its two variants is generally applicable to all languages for which WordNet exists, and for which the WordNet functions provided by Princeton University are available. In our first experiments, we choose English as a case study, and evaluate the model variants by pseudo-word disambiguation. Within the applications of our model (cf. Section 2.3), we build models for English and for German.

2.2.1 Experiment Setup

The first experiments were meant to explore the general potential of our approach, and to test and compare the various parameters the models provide. All models in this experiment series rely on corpus data from the British National Corpus (BNC, 1995). The training and test data were obtained by parsing the BNC corpus with a simple PCFG grammar (Carroll and Rooth, 1998) and extracting tuples containing verbs, frames and argument nouns, consisting of the tuple frequency, a verb, a frame, and a noun for each of the frame's slots. Here are some example tuples:

```
54      begin subj work
21      expect subj:to one find
22      hear subj:obj lord prayer
```

```
21      occur subj:pp-in message area
21      remain subj:ap question unanswered
```

We took only active clauses into account, and disregarded auxiliary and modal verbs as well as particle verbs, leaving a total of 4,852,371 Viterbi parses. Those input tuples were then divided into 90% training data and 10% test data, providing 4,367,130 training tuples (over 2,769,804 types), and 485,241 test tuples (over 368,103 types).

As we wanted to train and assess our verb class model under various conditions, we used different fractions of the training data in different training regimes. Due to time and memory constraints, we only used those training tuples that appeared at least twice, i.e., with at least two tokens. But for comparison reasons we also trained a selection of models including input tuples with a frequency of 1. Furthermore, we disregarded tuples with personal pronoun arguments; they are not represented in WordNet, and even if they are added (e.g., to general concepts such as *person, entity*) they have a rather destructive effect. We eliminated tuples with nouns which are not represented in WordNet. Since we have no smoothing for unknown verbs, we restricted the test data to verbs that also occurred in the training data.

Most importantly for the number of training tuples, we considered two subsets of 10 and 20 subcategorisation frames. The frame types were chosen according to their overall frequency in the training data. Table 2.1 shows the 20 most frequent frames ordered by frequency. A frame lists its arguments, separated by a colon ':'.

When relying on theses 10/20 subcategorisation frames, plus including the above preferences, we are left with 51,569/55,980 training tuples, respectively. The overall number of training tuple types is therefore much smaller than the generally available data. The corresponding numbers including tuples with a frequency of one are 671,461 and 815,553.

In addition to the above parameters, the experiments varied the number of clusters, using 20, 50, and in some non-memory-restricted cases also 100 clusters. Concerning the iterations over the training tuples, we used up to 50 iterations, and output the model probabilities after each 5th iteration, in order to check how the model developed within the training. The simplest model was even trained over 200 iterations, to get a more complete picture. Furthermore, we tried different seeds to initialise the random number generator for the initial probabilities in the EM training, using 10 different seed values. Last but not least, EPS provides a variant called vp that assumes for the sake of the computation of the model length that the probabilities (whose Maximum Likelihood Estimate is given by $\frac{f}{N}$) are stored with $-0.5\ log\ N$ bits rather than a fixed 32 bit precision (which is the default).

2.2.2 Pseudo-Word Disambiguation

Pseudo-word disambiguation is a popular evaluation method for selectional preference models. Its origins trace back to word sense disambiguation (WSD) where an efficient evaluation method

subj:obj
subj
subj:ap
subj:to
subj:obj:obj
subj:obj:pp-in
subj:adv
subj:pp-in
subj:vbase
subj:that
subj:pp-to
subj:vger
subj:s
subj:obj:to
subj:obj:adv
subj:pp-on
subj:obj:pp-for
subj:obj:pp-to
subj:pp-with
subj:pp-at

Table 2.1: 20 most frequent frames.

was desired that does not rely on hand-annotated test data. Thus, Schütze (1992) and Gale et al. (1992) independently developed an evaluation where a WSD algorithm was presented with a sentence in which all instances of a specific word are replaced by a **confounder**. The pair of the original word and its replacement is called a **pseudo-word**. The method was later adapted for selectional preferences by Dagan et al. (1999). Here, either the predicate or one of its arguments are replaced with a confounder. For example, if *eat/fly* is a pseudo-word, all occurrences of *eat* are replaced by *fly*. The verb-argument pairs

(2.1) *eat apple* and

(2.2) *fly apple*

are an example for such a replacement. Using this test set, a selectional preference model can be evaluated by presenting it with both the original sequence and the sequence that contains the confounder. The accuracy of an algorithm is the number of times the original is preferred by it in relation to the size of the test set. We express this through the following equation:

$$A = \frac{\text{number of correct choices} + 0.5 \times \text{number of ties}}{\text{number of tuples in test set}}$$

The main benefit of pseudo-word disambiguation is that no manual annotation is needed. The test data can be generated automatically under the condition that a list of pseudo-words can can be provided. Such a list can be obtained in various ways. As Chambers and Jurafsky (2010) suggests, a nearest frequency approach is the fairest way of choosing them.

First, we count each verb's occurrence in the corpus. We then sort the resulting list of verbs by frequency. Finally, we generate pseudo-words by assigning each verb the next frequent one as its confounder. The most frequent word does not receive a confounder that way and is thus excluded from the experiment.

Chambers and Jurafsky (2010) additionally notes that the results of pseudo-word disambiguation experiments vary depending on whether unseen data is used. For data that occurred in the training set, a simple baseline of conditional probabilities can be sufficient to beat more complicated selectional preference models. In our experiment, none of the test data was seen during training. Also, as suggested by the authors, no verbs were excluded from the test set.

Experimental Setup

For our experiments, the BNC training and test data as described in Section 2.2.1 was used.

The following parameters of the predicate argument clustering model are evaluated: (i) the number of iterations, (ii) the number of clusters, (iii) the MDL parameter w, (iv) the effect of the exclusion of tuples with frequency 1 in the training data, (v) the exclusion of infrequent frames, (vii) the use of the vp option, and (viii) the seed value for the random initialisation of the EM training.

We compare some of our models with corresponding latent semantic clustering (LSC) models and a simple frequency based baseline model. This models tries to identify the original tuple by taking only the verb and its confounder (v and v') and the subcategorisation frame (f) into account. Based on this data, the original is identified according to the algorithm in Figure 2.2.

if $c(\langle v, f\rangle) > 0$ or $c(\langle v', f\rangle) > 0$ **then**
  if $c(\langle v, f\rangle) > c(\langle v', f\rangle)$ **then**
    return v
  else
    return v'
  end if
else if $c(v') > c(v)$ **then**
  return v'
else
  return v
end if

Figure 2.2: Baseline algorithm based on verbs subcategorisation frames.

Results

Our main experiments in which all available methods are compared, the following parameters are tested:

- 10 and 20 frames
- 20, 50, and 100 clusters
- the vp option for EPS
- MDL weights $w = 1$ and $w = 0.5$
- minimum training data thresholds of 1 and 2

In the experiments with a frequency threshold of 1, the EPS-b model with 100 clusters outperforms all other models including the baseline when using 10 frames. With 20 frames, the baseline is equal to the EPS models. When removing items with a frequency of 1 from the training data, the baseline clearly outperforms all other models.

method	min_freq	frames	clusters	vp	5	10	15	20	25	30	35	40	45	50
LSC	1	10	20	–	70.34	73.93	75.10	74.90	75.28	75.53	75.79	75.92	76.05	75.85
NPS	1	10	20	–	54.11	54.66	62.30	65.98	66.99	68.00	68.54	69.01	69.01	50.00
EPS-a	1	10	20	no	58.11	69.56	72.68	74.55	75.62	75.54	76.22	76.64	76.81	76.84
EPS-a	1	10	20	yes	58.08	69.41	73.84	76.52	76.90	77.74	77.80	77.56	77.53	77.71
EPS-b	1	10	20	no	58.16	69.56	74.43	76.46	77.53	77.47	77.83	77.86	78.03	77.94
EPS-b	1	10	20	yes	58.22	69.77	75.06	77.32	77.89	78.66	78.66	78.78	78.60	78.78
LSC	1	10	50	–	71.67	75.30	74.99	75.73	75.34	75.64	75.76	75.58	75.58	75.33
NPS	1	10	50	–	54.19	54.96	62.85	67.80	69.96	71.15	70.62	71.32	71.42	71.95
EPS-a	1	10	50	no	58.58	69.59	73.16	75.18	75.65	76.01	76.61	76.70	77.44	77.53
EPS-a	1	10	50	yes	58.22	69.47	74.43	77.80	79.13	78.84	79.64	79.73	79.67	79.94
EPS-b	1	10	50	no	58.46	69.32	73.25	76.07	77.05	77.77	78.87	79.02	79.40	79.31
EPS-b	1	10	50	yes	58.34	69.85	75.27	78.09	79.70	80.26	80.06	80.06	79.97	80.12
EPS-a	1	10	100	no	58.37	68.81	72.83	75.15	76.07	76.01	75.98	75.89	75.86	76.49
EPS-a	1	10	100	yes	58.70	69.41	73.63	76.04	78.42	79.10	79.52	79.22	79.49	79.82
EPS-b	1	10	100	no	58.46	67.89	73.30	75.86	77.83	78.27	78.45	78.88	79.02	50.46
EPS-b	1	10	100	yes	58.61	69.32	74.38	78.06	79.70	**80.77**	80.80	80.41	80.00	79.82
Baseline					78.25									
LSC	1	20	20	–	71.61	74.28	74.63	74.16	73.90	74.28	74.74	74.78	74.63	74.59
NPS	1	20	20	–	50.46	54.85	60.95	63.07	64.22	65.98	66.30	0.00	67.27	67.17
EPS-a	1	20	20	no	52.33	71.59	74.51	75.19	76.35	77.22	77.73	77.57	77.71	78.06
EPS-a	1	20	20	yes	52.58	70.45	73.92	76.38	77.54	78.33	78.27	78.49	78.87	78.92
EPS-b	1	20	20	no	52.76	71.29	74.11	75.73	76.84	78.00	78.63	78.46	78.35	78.52
EPS-b	1	20	20	yes	52.71	69.94	74.46	76.35	77.11	77.71	77.95	78.49	78.52	78.44
EPS-a	1	20	50	no	52.38	71.19	75.49	77.16	78.33	79.22	79.36	79.57	79.46	79.63
EPS-a	1	20	50	yes	52.68	71.29	76.52	78.60	79.65	80.76	80.65	80.98	81.06	81.09
EPS-b	1	20	50	no	52.76	71.29	75.38	77.46	78.87	79.65	79.63	80.03	80.01	80.03
EPS-b	1	20	50	yes	52.62	71.35	76.14	78.98	80.19	80.65	80.65	80.65	80.57	80.87
EPS-a	1	20	100	no	52.30	71.89	75.97	77.68	78.95	79.03	79.49	79.76	79.76	79.71
EPS-a	1	20	100	yes	52.76	72.35	76.57	78.57	80.01	80.17	80.57	79.84	80.09	79.90
EPS-b	1	20	100	no	52.57	71.10	75.84	78.84	79.06	79.46	79.84	80.14	80.52	80.68
EPS-b	1	20	100	yes	52.84	72.40	76.79	79.30	81.14	81.49	81.63	81.41	80.90	81.20
Baseline					**81.67**									

Table 2.2: Accuracy for models including tuples with frequency one.

method	min_freq	frames	clusters	vp	5	10	15	20	25	30	35	40	45	50
LSC	2	10	20	–	59.89	60.90	60.35	60.41	60.22	59.91	60.22	59.97	59.94	59.86
NPS	2	10	20	–	51.45	51.80	51.45	53.22	52.21	52.24	54.42	54.13	54.16	54.79
EPS-a	2	10	20	no	48.58	57.25	57.90	58.69	59.05	58.26	58.56	58.52	58.16	58.20
EPS-a	2	10	20	yes	48.77	56.00	58.43	59.15	59.80	59.51	59.96	59.87	59.54	59.57
EPS-b	2	10	20	no	48.64	56.43	57.64	58.52	58.59	59.47	58.49	58.26	58.72	58.49
EPS-b	2	10	20	yes	48.71	55.87	57.54	58.46	57.84	57.74	57.58	57.13	56.95	57.18
LSC	2	10	50	–	59.89	59.95	59.09	59.12	59.48	59.74	59.61	59.71	59.73	59.64
NPS	2	10	50	–	51.73	52.30	50.28	52.46	51.89	53.72	53.06	52.96	53.06	50.00
EPS-a	2	10	50	no	48.61	56.00	56.95	57.64	58.13	58.43	57.87	58.20	58.10	58.23
EPS-a	2	10	50	yes	48.67	55.97	57.41	59.24	60.13	60.59	60.06	60.42	59.51	59.54
EPS-b	2	10	50	no	49.30	56.17	58.36	58.85	58.38	58.30	58.26	58.80	58.85	59.65
EPS-b	2	10	50	yes	48.41	56.20	57.25	58.26	58.49	58.16	58.10	58.36	58.00	57.71
EPS-a	2	10	100	no	48.45	56.07	56.59	57.08	58.23	58.16	58.46	57.25	57.58	57.72
EPS-a	2	10	100	yes	48.31	55.42	57.74	59.38	59.67	58.92	58.59	57.80	57.97	58.49
EPS-b	2	10	100	no	48.54	55.81	57.44	59.06	59.33	59.10	58.74	58.74	58.97	58.59
EPS-b	2	10	100	yes	48.35	56.20	58.13	58.77	58.08	58.51	58.92	59.47	59.44	59.02
Baseline					**64.08**									
LSC	2	20	20	–	60.03	59.29	59.14	59.35	59.16	59.03	58.96	58.87	58.80	58.84
NPS	2	20	20	–	50.19	49.58	51.17	50.37	50.43	50.69	50.12	49.88	49.88	50.00
EPS-a	2	20	20	no	48.93	55.31	55.59	54.04	54.37	53.25	52.59	52.44	52.29	53.28
EPS-a	2	20	20	yes	49.41	54.50	54.71	53.83	53.65	52.74	52.80	53.16	52.98	52.80
EPS-b	2	20	20	no	48.93	54.86	55.53	55.77	55.36	54.89	54.16	54.47	53.86	53.72
EPS-b	2	20	20	yes	49.11	54.59	54.30	53.57	53.41	53.53	53.38	53.16	53.04	53.04
EPS-a	2	20	50	no	49.41	53.65	54.62	53.50	54.07	54.07	53.77	53.03	53.15	53.06
EPS-a	2	20	50	yes	49.26	54.22	54.28	54.04	52.98	53.35	54.37	54.59	54.63	54.19
EPS-b	2	20	50	no	49.23	54.77	54.83	54.25	54.33	53.72	53.78	53.33	53.47	53.44
EPS-b	2	20	50	yes	48.93	54.74	54.00	54.07	53.51	53.60	53.32	52.80	52.35	52.19
EPS-a	2	20	100	no	49.14	53.95	55.43	54.50	54.19	54.01	53.94	53.65	53.78	53.53
EPS-a	2	20	100	yes	49.05	54.80	54.74	54.25	54.16	54.28	53.30	53.00	52.72	52.77
EPS-b	2	20	100	no	49.26	55.43	55.03	54.09	54.10	54.50	54.00	53.48	53.63	53.00
EPS-b	2	20	100	yes	48.93	54.83	54.44	54.04	53.86	53.62	53.81	53.89	53.92	53.59
Baseline					**65.55**									

Table 2.3: Accuracy for models excluding tuples with frequency one.

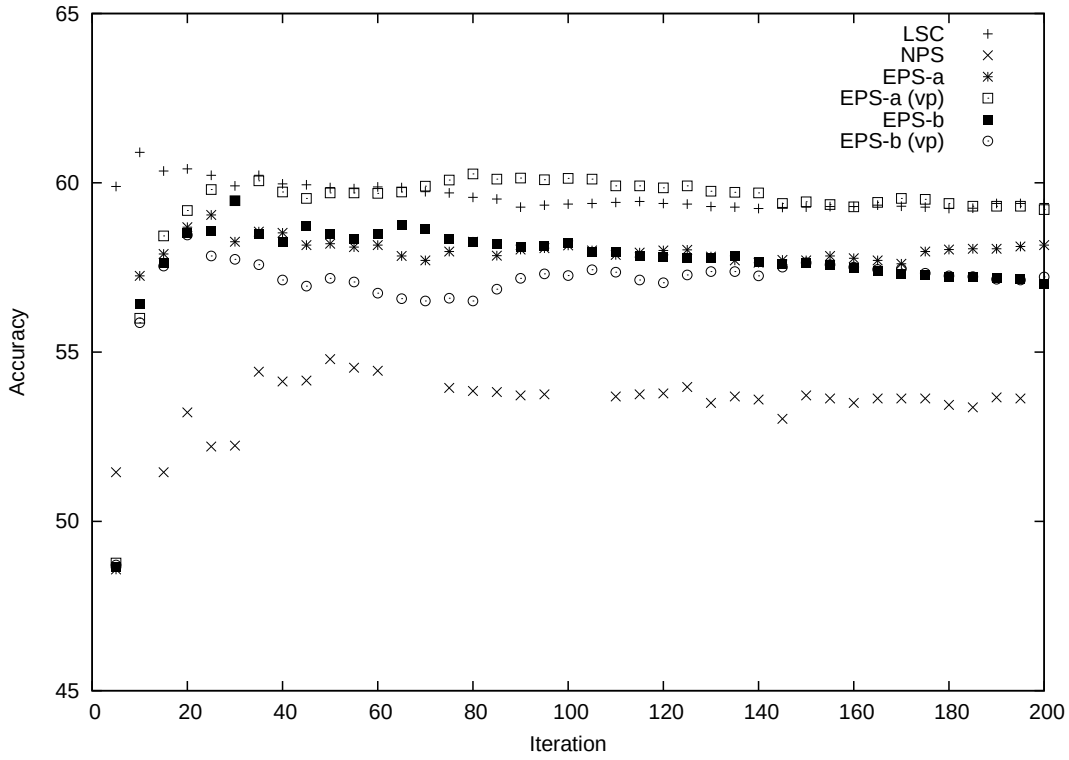

Figure 2.3: Accuracy for models with 200 iterations.

vp option. Using the vp training variation (cf. Section 2.2.1), the accuracies differ by up to about one percentile from the non-vp models. This effect has been observed across all experiments.

Number of iterations. Our standard experiments were conducted with up to 50 training iterations. In order to determine whether a higher number of iterations could improve the accuracy, we trained one model (10 frames, 20 clusters) using 200 iterations. Figure 2.2.2 shows the results for these models.

The accuracy for LSC peaks early at around 10 iterations, NPS performs best at around 45 iterations.

Number of clusters. We compared the performance of LSC, NPS, and EPS models with 20, 50, 100, 200, and 300 clusters to check whether a higher number of clusters which could theoretically lead to a better representation of the training data is beneficial for pseudo-word disambiguation. Figure 2.2.2 shows the maximum accuracies for each number of clusters. We can see that accuracy of NPS/EPS decreases when using 200 or more clusters. This suggests that a high

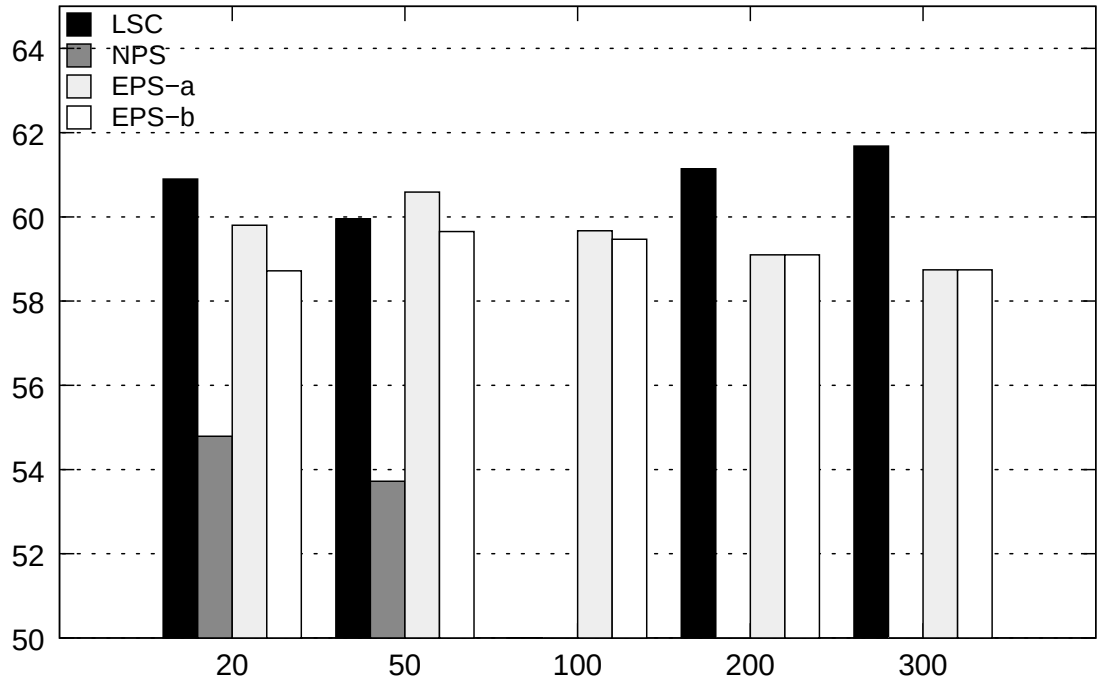

Figure 2.4: Best models for different numbers of clusters.

number of clusters leads to a bias towards the training data which has already been observed in the pseudo-word disambiguation evaluation of LSC by Rooth et al. (1999). The accuracies of LSC however still rise when using more clusters. We suspect that since we used a much larger amount in our experiments, more clusters are necessary to fit it and thus overfitting of LSC might only occur for an even higher number of clusters.

Data frequency.

MDL parameter. The MDL parameter w is responsible for the balance between the model (ML) and the description length (DL). If the description length gets a higher weight, the selection preference models grow to be more detailed during training since this leads to the training data being represented better. In the following experiments we compare two settings for w. $w = 1$ is the standard setting in which the model and description length are weighted equally, and $w = 0.5$ weights the description length higher. A lower value for w leads to slightly higher accuracy (80.71%) when compared with the standard setting (79.31%). However, this change leads to significantly higher training times which is why we did not use it for all experiments.

Set of frames. To test the model's capability of handling sparse data, we included less frequent subcategorisation frames in the model. As shown in Tables 2.2.2 and 2.2.2, there is no significant

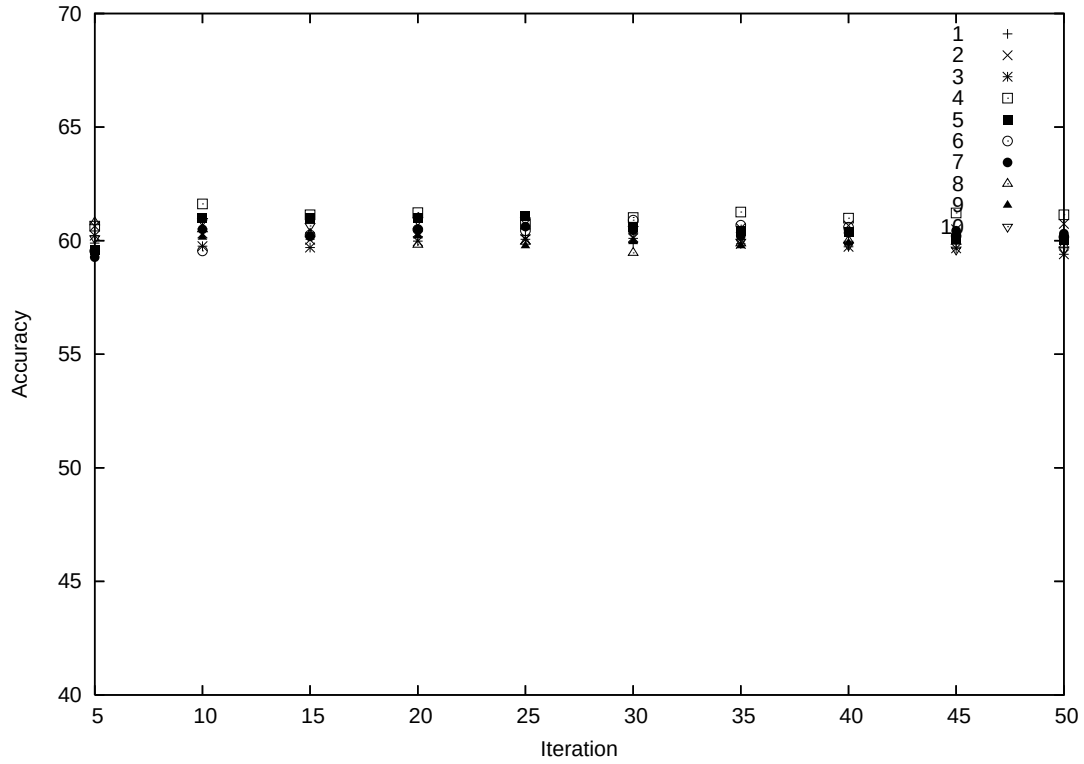

Figure 2.5: Accuracies of ten models with different seed values.

change in accuracy when using the 20 most frequent frames instead of using only 10.

Seed values. The seed value of the model determines the random initialisation values of its parameters. Changing the seed value can give us insights about how stable the training process is. As Table/Figure 2.2.2 shows, changing the seed value does not influence accuracy significantly.

Examples

We will investigate the experimental outcome more closely by looking at selected examples from the test data. All example tuples will be presented in the following format:

Probability	Verb	Frame	Fillers		
5.33571e-14	pronounce	subj:obj:obj2	government	sale	success

As showed by Chambers and Jurafsky (2010), the results pseudo-word disambiguation depend heavily on the choice of the pseudo-words. In our case, this problem can be illustrated by the following example tuples, each pairs of real words and confounders:

8.51234e-09	bend	subj:adv	head	forward
4.36423e-16	accompany	subj:adv	head	forward
4.02534e-12	glow	subj:ap	ruby	red
4.62002e-15	protect	subj:ap	ruby	red
1.23352e-12	contrast	subj:obj	pupil	manuscript
5.84823e-13	park	subj:obj	pupil	manuscript
8.79867e-09	debate	subj	consideration	
2.44797e-09	contest	subj	consideration	
3.88519e-10	gleam	subj:pp-with	eye	hatred
1.46639e-10	groan	subj:pp-with	eye	hatred
9.16453e-11	print	subj:obj	computer	memory
2.57217e-11	feed	subj:obj	computer	memory

While many pseudo-words consist of words that are semantically distant (e.g. *bend/accompany*, *glow/protect*, and *contrast/park*), there are examples in the test data for when the selection method produces pairs where the semantic differences are small, for example *debate/contest*. In this pseudo-word, two verbs with equivalent senses are combined. *print* and *feed* both can be used as technical terms from the printing domain which makes it more difficult to distinguish their correct uses. The two verbs in *gleam/groan* are related as well because they both describe actions where either sound or light is emitted from a source.

The strong influence of subcategorisation frames has been shown by the application of a frequency-based baseline method. When examining the data manually, cases like

1.4533e-11	assemble	subj:pp-in	audience	hall
4.65715e-14	contrast	subj:pp-in	audience	hall

make it clear that the correct use of some verbs can be easily determined by the frame chosen. *contrast* requires a *pp-with* instead of a *pp-in* and is thus unlikely to be the correct verb in this tuple.

3.40161e-15	charge	subj:obj:pp-for	farmer	premium	produce
1.45518e-20	flourish	subj:obj:pp-for	farmer	premium	produce

The less information is available, the less accurate the method becomes. This often is the case for intransitive verbs where only subject information is available:

1.16396e-08	confuse	subj	shadow
1.26925e-08	signify	subj	shadow

Conclusion

Our experiments show that pseudo-disambiguation is possible with our clustering approach. However, the simple baseline approach we introduced was still superior in most cases. This is a common outcome when using this evaluation method.

We are able to draw multiple conclusions from our experiments. In our overall experiments EPS is most effective when data that includes items with a frequency of 1 is used in training. Changes of the model parameters have an effect on the disambiguation accuracy. Increasing the number of clusters leads to small improvements of accuracy, using more iterations than about 50 however does not significantly change the results. Using the vp version of the model and producing more detailed selection preference models by changing the MDL parameter leads to sightly higher accuracies as well.

2.3 Applications

This section applies our clustering model in its variations to three tasks, selectional preference acquisition (Section 2.3.1), verb sense disambiguation (Section 2.3.2), and semi-supervised sense labelling (Section 2.3.3). Next to exploiting the general potential of the clustering model per se, the applications allow to compare the variations of the model, as well as to compare the model with LSC, the clustering model that is most similar to ours but does not incorporate selectional preferences (cf. Section 2.1.5). Model 1 (NPS) is henceforth refered to as *LSCpref*, model 2.1 (EPS-a) as *PAC-1*, and model 2.2 (EPS-b) as *PAC-2*, as these are the actual names of the implemented tools, cf. Chapter 3.

2.3.1 Selectional Preference Acquisition

Introduction

Predicates impose selectional restrictions on the realisation of their complements, as first illustrated by Chomsky (1957) through his famous example *"Colorless green ideas sleep furiously"*. Though the sentence is syntactically well-formed, it is not semantically meaningful, unless interpreted metaphorically. Compare also examples (1) and (2), where most people would agree that a chocolate cake is highly acceptable as the patient of the verb *bake*, but a stone is less typical (though, again, it might be subsumed by the context, e.g., as metaphorical when the baking result was unenjoyably hard).

(2.3) Elsa baked *a chocolate cake*.

(2.4) ?Elsa baked *a stone*.

Selectional preference acquisition is probably one of the most direct applications of our clustering approaches, which aim to model selectional preferences as part of the clustering information. In the following, we compare our clustering model 2 (EPS) to its simpler predecessor LSC as well as a distributional approach. The core part of the work has been published as Schulte im Walde (2010). All experiments were carried out for German, but can be transferred to other languages, given that sufficient corpus data is available to extract predicate–complement pairs, plus assuming a WordNet for our model.

Selectional Preference Models

Existing approaches to the automatic induction of selectional preferences have briefly been mentioned in Section 2.1.5. They fall into three categories. The majority of the approaches models selectional preferences by exploiting the hypernym/hyponym hierarchy in WordNet (Fellbaum, 1998). Relying on corpus-based *predicate–relation–noun* frequencies, they aim to find the optimal generalisation of the nouns as selectional preference characterisation with respect to the predicate and the predicate–noun relation. As the result, the selectional restrictions are expressed by WordNet classes, or sets over WordNet classes (most commonly a disjunctive set of classes represented by a *cut* through the hierarchy). Referring to the above example, having seen *coffee, tea, beer* and *wine* as direct objects of the verb *drink*, the hypernym *beverage* generalises over the seen nouns and thus represents a suitable WordNet label for the verb–object selectional preference. Approaches that fall into this category are Resnik (1997), Li and Abe (1998), Abney and Light (1999), Ciaramita and Johnson (2000), and Clark and Weir (2002).

An alternative to WordNet-based models are cluster-based models such as Pereira et al. (1993) and LSC (Rooth et al., 1999). Also relying on corpus-based *predicate–relation–noun* frequencies, cluster-based approaches represent selectional preferences by noun clusters that generalise over the seen nouns, without specific generalisation labels other than the cluster numbers. Two of our models (LSC and ESP) fall into this category, with the more complex one (i.e., EPS) refining the selectional preference description by WordNet categories.

Last but not least, Erk (2007) suggested a distributional, similarity-based model for selectional preferences that uses the corpus-based input data to first define a selectional preference representation, and then use vector-based similarity metrics to determine selectional preference scores for unseen nouns. Our second-order co-occurrence model is an instance of a distributional model, in many respects similar to Erk's model.

While WordNet-based approaches are attractive models of selectional preferences in that they explicitly provide preference categories, cluster-based and similarity-based approaches are attractive in that they are independent of such a manual resource which is not available for all languages and is costly to build.

In the following, we introduce our three selectional preference models.

A Second-Order Distributional Model According to the distributional hypothesis, the sum of contexts of a linguistic unit is a crucial indicator of the meaning of the linguistic unit (Firth, 1957; Harris, 1968). In this vein, we define a distributional approach to selectional preference induction that is both intuitive and cheap. The underlying idea is that selectional preferences of a predicate's complement are defined by the properties of the complement realisations. For example, a typical direct object of the verb *drink* is usually fluid, might be hot or cold, can be bought, might be bottled, etc. So –referring to this example– are adjectives that modify nouns, or verbs that subcategorise nouns salient properties to describe the selectional preferences of direct objects? The general question we ask is: what characterises the realisations of selectional preferences? We thus suggest a second-order co-occurrence model for selectional preferences: a predicate's restrictions to the semantic realisation of its complements are expressed through the properties of the complements.

The basis of the distributional approach are corpus-based co-occurrences of triples
$\langle predicate, relation, complement \rangle$,
i.e., joint frequencies of predicate–complement pairs with respect to a specific functional relation. Being of second-order co-occurrence, the model combines two types of co-occurrences: (1) Corpus-based joint frequencies $freq(p, r1, n)$ of predicates p and nouns n with respect to some functional relationship $r1$. These co-occurrences refer to the functional relationships whose selectional preferences we address. We concentrate on German verb–noun relationships $r1$, namely subjects, direct object, and pp objects. This choice was motivated by the language we work on (German), plus the available data for evaluation, cf. Section 3. The approach can easily be expanded to other predicates and relations, but in order to incorporate the latent semantic class model with WordNet generalisations, the complement choice is necessarily nouns. (2) Corpus-based joint frequencies $freq(n, r2, prop)$ of nouns n and noun properties $prop$ with respect to some functional relationship $r2$. These co-occurrences refer to the properties of the selectional preferences we address. We concentrate on modifying adjectives, subcategorising verbs (for direct object and pp object), and subcategorising prepositions, because these properties were expected to shed light on complementary semantic properties of the nouns (and thus the selectional preference descriptions). We tested the properties by themselves, and also in combinations. The set of properties can easily be enlarged, as the experiments will demonstrate. The joint frequencies were estimated on approx. 560 million words from the German web corpus *deWaC* (Baroni and Kilgarriff, 2006), after the corpus was preprocessed by the Tree Tagger (Schmid, 1994) and by a dependency parser (Schiehlen, 2003).

The distributional model comprises two parts: (1) the selectional preference description with respect to a specific verb–noun relationship, i.e., the second-order properties of the relationships, and (2) the selectional preference fit of a specific noun with respect to the verb–noun relationship. Part (1) is a simple scoring that combines the two types of corpus-based joint frequencies, $freq(p, r1, n)$ and $freq(n, r2, prop)$, cf. Equations (2.5) to (2.8). The second-order selectional preference of the verb–noun relationship $r1$ is represented by the joint noun–property corpus frequencies across the nominal complements, cf. Equation (2.5) for the most basic version. I.e., the feature vector of the predicate is a union of the properties of the nouns. For example, if the predicate is the verb *drink*, the verb–noun relation is a direct object, and the property is adjectives

that modify nouns; then, the verb's selectional restrictions are defined by an adjective feature vector, where the set of adjectives is the union of the adjectives modifying the nouns subcategorised by ***drink***. The feature values $score_1(drink, dir\_obj, adj)$ thus rely on the frequencies of all nouns that appeared as direct objects of ***drink***, $freq(drink, dir\_obj, n)$, and on the frequencies of the adjectives those nouns appeared with (not necessarily in the same context with the verb, $freq(n, n\_mod, adj)$). For example, if ***coffee*** appeared 50 times as direct object of ***drink***, and ***tea*** appeared 5 times, and if ***coffee*** was modified by the adjective ***hot*** 100 times and by ***fluid*** 30 times, and if ***tea*** was modified by ***hot*** 60 times and by ***fluid*** 15 times, then $score_1(drink, dir\_obj, hot) = 50 * 100 \ + \ 5 * 60 = 5,300$, and $score_1(drink, dir\_obj, fluid) = 50 * 30 \ + \ 5 * 15 = 1,575$. The scoring provided in Equation (2.5) is the most simplest, using raw corpus frequencies. Alternative versions rely on log-transformed frequencies (Equation (2.6)), probabilities (Equation (2.7)), and tf-idf values (Equation (2.8), where $tf\_idf(triple) = tf(triple) * idf(triple)$, with $tf(triple) = prob(triple)$, and $idf(triple) = log\frac{|p,r1|}{|p,r1,n|}$ for $r1$, and $idf(triple) = log\frac{|n,r2|}{|n,r2,prop|}$ for $r2$, i.e., determining the "inverse document frequency" of nouns ($r1$) or properties ($r2$) by incorporating the number of different predicates ($r1$) or nouns ($r2$) a noun ($r1$) or property ($r2$) occurred with).

Tables 2.4 to 2.6 present examples of second-order properties, for the direct objects of the verb *backen* 'bake' with adjective properties, *anbraten* 'fry' with verb properties, and *abschmecken* 'taste' with preposition properties, respectively. The tables list the eight most probable properties and also the eight most probable nominal realisations, according to some of the most successful distributional models. The information is rather for intuitive purposes; therefore, the system scores are omitted. The prepositions in Table 2.6 are the most difficult to grasp intuitively, but at the same time the most successful system features, as the results will demonstrate.

$$score_1(p, r1, prop) = \sum_{n \in (p,r1)} freq(p, r1, n) \ * \ freq(n, r2, prop) \tag{2.5}$$

$$score_2(p, r1, prop) = \sum_{n \in (p,r1)} log(freq(p, r1, n)) \ * \ log(freq(n, r2, prop)) \tag{2.6}$$

$$score_3(p, r1, prop) = \sum_{n \in (p,r1)} prob(p, r1, n) \ * \ prob(n, r2, prop) \tag{2.7}$$

$$score_4(p, r1, prop) = \sum_{n \in (p,r1)} tf\_idf(p, r1, n) \ * \ tf\_idf(n, r2, prop) \tag{2.8}$$

Figure 2.6: Second-order selectional preference description.

As mentioned before, the resulting selectional preference descriptions are predicate vectors over complement properties. In part (2), the natural fit of a specific noun can then be specified by standard distributional similarity measures, comparing a specific noun's contribution to the overall preference: In order to determine the selectional preference for a specific (seen or unseen) noun, we calculate the vector-based similarity between the predicate's preference vector and the specific noun's vector. The measures to calculate the similarities and thus the natural fit of a specific

noun to a selectional preference description can be varied. We experimented with four standard measures that were expected to provide different perspectives on the selectional preference fit, due to their mathematical nature: the cosine of the vector's angle (a standard measure in linear algebra), the skew divergence, an information-theoretic measure and variant of the Kullback-Leibler divergence (Lee, 2001), Kendall's τ, a measure for rank correlation (Hatzivassiloglou and McKeown, 1993), and jaccard, a binary distance measure (Manning and Schütze, 1999).

Our method is similar to Erk's approach who also used complements' corpus-based properties to describe selectional preferences. We addressed the task from a different direction, though, and the result is a simplified version of her approach. The models with a single nominal property are specific cases of her model, and only the models with combined nominal properties come close to a general distributional description. Furthermore, our goal is different from hers in that we were interested in the contributions of the various properties, in addition to determining the natural fit of nouns to selectional preferences.

Properties: adjectives		Example realisations	
frisch	'fresh'	Keks	'cookie'
lecker	'delicious'	Brötchen	'roll'
klein	'small'	Torte	'tart'
trocken	'dry'	Kuchen	'cake'
süß	'sweet'	Brot	'bread'
warm	'warm'	Pizza	'pizza'
fett	'fat'	Waffel	'waffle'
eingeweicht	'soaked'	Pfannkuchen	'pancake'

Table 2.4: Direct objects of *backen* 'bake'.

Properties: verbs$_{NPacc}$		Example realisations	
schälen	'peel'	Champignon	'mushroom'
schneiden	'cut'	Zwiebel	'onion'
essen	'eat'	Kartoffel	'potato'
zugeben	'add'	Gemüse	'vegetable'
anschwitzen	'sweat'	Knoblauch	'garlic'
pellen	'peel'	Hackfleisch	'minced meat'
riechen	'smell'	Roulade	'roulade'
waschen	'clean'	Keule	'haunch'

Table 2.5: Direct objects of *anbraten* 'fry'.

Properties: prepositions		Example realisations	
mit	'with'	Soße	'sauce'
in	'in'	Salat	'salad'
für	'for'	Brühe	'stock'
zu	'for'	Gemüse	'vegetables'
von	'from'	Eintopf	'stew'
unter	'under'	Suppe	'soup'
auf	'on'	Püree	'puree'
als	'as'	Essen	'food'

Table 2.6: Direct objects of *abschmecken* 'taste'.

Latent Semantic Classes The Latent Semantic approach has previously been applied to model the selectional dependencies between two sets of words participating in a grammatical relationship (Rooth et al., 1999). LSC was chosen because the clusters can be considered as generalisations over the members of the two inter-dependent dimensions. The LSC approach therefore fits selectional preferences, by generalising over seen and unseen lexical items.

Our experiments with LSC rely on the same corpus data as the distributional model; we used the same verb–subject, verb–direct-object, and verb–pp-object data. We trained three LSC models, one for each functional relation, and a forth model that contained all relations, using a relation marker at the verb (e.g., replacing the verb *backen* with *backen-subj*) to distinguish between the relations. The resulting analyses were used to calculate the probabilities of verb–noun pairs as the natural fit of the nouns to the selectional preferences the clusters incorporate. The training parameters were varied, producing cluster analyses with 20, 50, 100, 200, and 500 clusters, over 50 and 100 iterations.

Table 2.7 presents an LSC example of a cluster containing verbs and their direct objects, as taken from a 100-cluster analysis. The left-hand column contains the most probable predicates within this cluster; the right-hand column contains the most probable nouns within this cluster. The nouns are assumed to represent the selectional preferences of the direct objects of the verb dimension.

cluster, prob(c) = 0.015 (range: 0.004-0.035)			
entwickeln	'develop'	Konzept	'concept'
vorstellen	'introduce'	Angebot	'offer'
erarbeiten	'work out'	Vorschlag	'suggestion'
geben	'give'	Idee	'idea'
umsetzen	'realise'	Projekt	'project'
ansehen	'look at'	Plan	'plan'
erstellen	'create'	Programm	'program'
präsentieren	'present'	Strategie	'strategy'
diskutieren	'discuss'	Modell	'model'
darstellen	'demonstrate'	Lösung	'solution'

Table 2.7: Example LSC cluster.

Latent Semantic Classes integrating Selectional Preferences Our experiments with PAC rely on the same corpus data as the other two models. We used and compared both clustering model 2.1 (EPS-a) and 2.2 (EPS-b). In both cases, we used the same verb–subject, verb–direct-object, and verb–pp-object data, and we trained four PAC models, one for each functional relation, and one for all data in one model (as PAC incorporates frame types and thus distinguishes between functional relations). As for LSC, we used the resulting analyses to calculate the probabilities of verb–noun pairs as the natural fit of the nouns to the selectional preferences the clusters

incorporate. The training parameters were varied as for LSC, producing cluster analyses with 20, 50, 100, 200, and 500 clusters, over 50 and 100 iterations.

Table 2.8 presents a PAC example of a cluster containing verbs and their direct objects, as taken from a 20-cluster analysis. The left-hand column contains the most probable predicates within this cluster; the right-hand column contains a selection of the most probably WordNet classes from different hierarchical levels. The extensive WordNet hierarchical structure that is part of the second cluster dimension is omitted for space and clarity reasons.

cluster, prob(c) = 0.069 (range: 0.014-0.085)			
leisten	'perform'	Geschehen	'event'
geben	'give'	Aktivität	'activity'
fordern	'demand'	Veränderung	'change'
bedeuten	'mean'	Handlungssequenz	'action sequence'
ermöglichen	'enable'	Realisierung	'realisation'
verhindern	'prevent'	Anschlag	'attack'
feiern	'celebrate'	Straftat	'criminal act'
darstellen	'demonstrate'	Gerichtsverfahren	'lawsuit'
bringen	'bring'	Verbesserung	'improvement'
vornehmen	'carry out'	Optimierung	'optimisation'

Table 2.8: Example PAC cluster.

Evaluation

The three selectional preference approaches were evaluated against human judgements on German verb–noun pairs. The judgements had been collected by Brockmann and Lapata (2003)[6] whose study compared the WordNet-based selectional preference approaches by Resnik (1997), Li and Abe (1998), and Clark and Weir (2002), plus two distributional models relying on co-occurrence frequency and conditional probability. The human data contains 90 verb–noun pairs, with 30 pairs each for subjects, direct objects and pp objects, and each of the 30 pairs contains 10 different verbs with 3 different nouns. Verbs and nouns were chosen randomly; furthermore, the noun choice was done in accordance with three frequency bands of the verb–relation–noun triples. The participants in the study were asked to provide selectional preference scores for the 90 verb–noun pairs; the scores were then normalised to a common scale, and transformed by taking the decadic logarithm $log10$.

Brockmann and Lapata used the human judgements to compare the above-mentioned selectional preference approaches. Each model provided selectional preference scores for the 90 verb–noun pairs, the system scores were transformed by taking the decadic logarithm and then correlated

[6]Thanks to Carsten Brockmann for providing the judgement scores to us.

against the human judgement scores by linear regression. Brockmann and Lapata found that all five models were significantly correlated with the human judgements, but inter-subject agreement was consistently higher than the correlations. Furthermore, no model performed best; different methods were suited for different functional relations. A combination of the models by multiple linear regression outperformed the individual models.

By using the same gold standard data and the same computations as Brockmann and Lapata, we can compare not only our models against each other, but also compare our results to theirs. We therefore calculated system scores for the 90 verb–noun pairs (which had previously been removed from the training data) with respect to our three approaches. As Brockmann and Lapata, we also transformed the system scores by taking the decadic logarithm, before performing the linear regression with their $log10$ human judgements. In comparison, however, we also correlated the original system scores against the human judgements back-transformed by the $log10$ reverse function. The latter procedure seemed reasonable, as we did not agree with a general $log10$ transformation without knowledge about the underlying data distribution.[7]

Furthermore, we added a second type of evaluation, and compared the approaches using the Spearman rank-order correlation coefficient (henceforth: *ranking*). This correlation is a non-parametric statistical test that measures the association between two variables that are ranked in two ordered series. The ranking seemed reasonable, as it looked at the evaluation from a different perspective, assessing how well the systems can distinguish fine-grained rank-order differences across the gold standard pairs.

The baselines of the experiments were calculated by correlating the joint corpus-based predicate–noun frequencies of the subjects, direct objects and pp-objects with the human judgements (also by linear regression, and by ranking). The upper bound of the approaches is referred to as the inter-subject agreement (*isa*) on the selectional preference judgements, as calculated by Brockmann and Lapata, henceforth *BL*.

Results

Tables 2.9 to 2.12 present an extract of the results of the distributional approach, and the LSC and PAC experiments. All of the results refer to the evaluation by linear regression, where the system scores were not transformed by the decadic logarithm (and, accordingly, correlated with the back-transformed judgements); the results with respect to ranking will be described below. In each table, our best results per column are printed in bold font. The overall best results per relation are in addition printed in blue, and marked by the significance levels $*p \leq .05$, $**p \leq .01$, and $***p \leq .001$, if applicable. The BL results in Table 2.9 refer to the best results achieved in the Brockmann/Lapata comparison, and provide the respective system in brackets.

[7] If the data is normally distributed without transformation, then it needs no transformation to go into a linear regression; if the data is normally distributed after the transformation, then a transformation is reasonable. In any case, the transformation will change the linear regression, as a logarithm imposes a shape on the scores that influences the linearity. The degree of the change depends on the scale of the scores.

The baseline and upper bound values are only listed in Table 2.9 but refer to all linear regression experiments in the three tables. The frequency baseline correlated the joint predicate–noun frequencies against the back-transformed human judgement scores; the $log10(f)$ baseline correlated the frequencies transformed by the decadic logarithm against the BL judgement scores; the BL baseline is taken from their paper and refers to $log10$-transformed frequencies correlated against the $log10$ BL judgement scores.

The distributional results list the *cosine* scores as the measure of selectional preference fit, because it provided the overall best scores. The rows refer to the second-order properties, and the columns to the second-order selectional preference description, cf. Figure 2.6. As mentioned before, we used adjectives, verbs, and prepositions as second-order properties; furthermore, we enlarged and unified the property sets: $v + vp$ adds verb–preposition pairs (subcategorising for the respective nouns), $v+vp+adj$ adds adjectives to this set, and $v+vp+adj+prep$ further adds the prepositions. A number of things are striking in Table 2.9: (1) Not only with respect to the cosine results but also in more general, the prepositions by themselves, or the union of second-order properties $v + vp + adj + prep$ are in many cases the most successful properties. On the one hand, we can conclude that prepositions are a powerful indicator of selectional preference properties; on the other hand, our largest set of properties comes close to a general distributional description without strong restrictions on the selection of properties, in the vein of Erk (2007), and the question is whether a less careful choice than the properties we provided would be as successful or even more successful. One could try, e.g., window information as a very crude choice. (2) The best results vary quite strongly with respect to the functional relation. Direct objects are modelled best, subjects are modelled worst. (3) Not only with respect to this table but in more general, the probability and tf-idf scores tended to outperform the frequency- and log(f)-based scores. Note that the best overall result in Table 2.9 is achieved by frequency, though. (4) Quite striking in the table are the large values of the baseline using the log-transformed predicate–noun frequencies, $.652/.559/.565/.574$ for *subj, dir obj, pp-obj, overall* with an upper bound for isa of $.790/.810/.820/.810$, cf. BL. The baseline is so high that it beats some of the best system (and system combination results) in BL ($.408/.611/.597/.400$), and some of our results ($.494/.713/.602/.517$). Furthermore, our baseline is much higher than BL's baseline (calculated identically, as far as we know): $.386/.360/.168/.301$. The only explanation for this is that the results differ because of the different underlying corpora, 560 million words of the *deWaC* vs. 179 million words of the German *Süddeutsche Zeitung* newspaper corpus. To be sure whether the size or the domain differences are the crucial ingredients, one would have to replicate our experiments on a portion of the *deWaC* comparable to BL's portion. We hypothesise, though, that the difference is rather due to the corpus domains, which should arguably provide different frequency counts for verb–noun pairs such as *reward a child*, or *clean the pavement*, whose German translations are among the gold standard pairs. The same reason applies to the fact that our results are all above those of BL's comparison. One would have to re-run the various systems on our data, in order to have a fair comparison. (5) As mentioned above, the cosine measure was the most useful for our purposes. The skew divergence and the jaccard binary measures were always clearly below the cosine-based scores. Only the results with Kendall's τ were in some cases similar to the cosine results; for subjects, τ could even beat the cosine, with a correlation of

**.532, using $v + vp$. (6) The cosine results when correlating the $log10$ system scores against the $log10$ judgements were quite below the ones in Table 2.9, confirming our intuition that a general $log10$ transformation and linear regression do not necessarily fit.

Table 2.10 presents the results for the LSC experiments. The first column for each relation refers to a linear regression of the probabilities of the verb–noun pairs and the back-transformed judgements; the second column refers to the correlation between the $log10$-transformed probabilities against the $log10$-transformed judgements. Although the best LSC correlations are also significant, all of them are below those of the simpler distributional model. Interestingly, though, the correlations based on the $log10$-transformed scores were in most cases above those without transformation. Concerning the number of clusters and training iterations, there is no clear tendency towards an optimal settings. The number of training iterations did not consistently improve the results, and neither did a smaller or larger number of clusters. When training used all relation information at the same time ($all\ func$), relying on relation markers at the verb (cf. Section 2.2), LSC performed better than after individual training on the relation data.

Tables 2.11 and 2.12 presents the results for the PAC experiments. Again, the first column for each relation refers to a linear correlation between the probabilities of the verb–noun pairs against the back-transformed judgements, and the second column refers to the correlation between the $log10$-transformed probabilities against the $log10$-transformed judgements. The PAC results vary quite strongly with respect to the underlying MDL approach: The EPS-b results are much better than the EPS-a results. The former even outperform most of the distributional results (i.e., all but the pp object preferences), while the latter are very poor with respect to pp objects, and otherwise similar or slightly worse than the distributional results. All PAC results (except for ESP-a on direct object preferences) are better than the LSC results. I.e., the generalisation over nouns in the verb–noun data by PAC in comparison to LSC obviously improves on selectional preference prediction. As for LSC, the correlations based on the $log10$-transformed scores were in most cases above those without transformation. Also similar is the fact that the number of clusters and iterations does not have a clear tendency towards selectional preference prediction.

The evaluation by the Spearman rank-order correlation coefficient provides similar results as the linear regression evaluation. The tables are omitted for space reasons. Again, the distributional model using the cosine is identified as the most successful selectional preference approach. In comparison to a baseline of $.903/.863/.928/.884$ (where the ranking according to the joint predicate–noun frequencies is correlated against the gold standard ranking), the cosine reaches best correlations of $.880/.938/.947/.879$ for subject, direct object, pp object and across relation selectional preferences. It thus beats the baseline in all cases but the subject. In comparison, the distributional model using the skew divergence achieves only correlations of $.758/.739/.773/.772$. LSC and PAC reach correlations of $.872/.872/.896/.873$ (LSC), $.882/.877./.795/.850$ (EPS-a) and $.904/.904/.880/.879$ (EPS-b). The results of LSC and EPS-a are therefore in most cases below those of the distributional model, and the results of EPS-b are similar to those of the distributional model.

The properties of the most successful distributional models, and the number of clusters and training iterations of the most successful cluster models are not the same as those in the linear

regression evaluation. Therefore, we cannot conclude about any general optimal settings of the models.

	SUBJ		DIR-OBJ		PP-OBJ		*all*	
	f	log(f)	f	log(f)	f	log(f)	f	log(f)
adj	.416	.373	.417	.261	.113	.220	.244	.156
verb	.456	.412	.271	.222	.176	.278	.201	.178
prep	.461	.345	.681	.263	.318	.393	.391	.272
v+vp	**.468**	.425	.345	.295	.344	.369	.295	.235
v+vp+adj	.420	.411	.388	.287	.235	.345	.285	.222
v+vp+adj+prep	.459	**.465**	***.713	**.328**	**.380**	**.476**	**.422**	**.359**
	prob	tf-idf	prob	tf-idf	prob	tf-idf	prob	tf-idf
adj	.430	.420	.352	.301	.339	.373	.309	.311
verb	**.494	.406	.285	.325	.242	.487	.273	.386
prep	.443	**.487**	.625	**.680**	.554	***.602	.481	**.516**
v+vp	.479	.387	.333	.290	.476	.564	.345	.401
v+vp+adj	.435	.383	.402	.307	.401	.478	.345	.364
v+vp+adj+prep	.465	.437	**.705**	.428	**.599**	.581	***.517	.455
BL	*.408 (Resnik)		***.611 (Clark/Weir)		***.597 (Clark/Weir)		***.400 (comb)	
baselines & upper bound								
baseline: f	.274		.343		.384		.313	
baseline: log10(f)	.652		.559		.565		.574	
baseline: BL	.386		.360		.168		.301	
isa	.790		.810		.820		.810	

Table 2.9: Distributional results.

Conclusions

We compared our clustering model against the simpler LSC, and a distributional model. Quantitative and qualitative analyses of the approaches demonstrated that the latest version of our clustering model is the most successful model, outperforming an earlier version as well as LSC, and in most cases also the distributional results. Even the best models, though, did not always beat the powerful frequency baseline.

	SUBJ		DIR-OBJ		PP-OBJ		*all*		*all-func*	
50 training iterations										
20	.253	*.450	.016	.282	.181	.295	.033	.338	.118	.383
50	**.332**	.382	.074	.424	.117	.061	.172	.240	.185	***.453
100	.202	.222	**.313**	.483	.234	.141	.203	.235	.081	.379
200	.310	.308	.285	.469	.243	.189	**.216**	.275	**.226**	.332
500	.261	.210	.258	.393	.318	.189	.157	.242	.155	.339
100 training iterations										
20	.249	.165	.061	.386	.149	.352	.064	.266	.096	.362
50	.320	.317	.184	.420	.069	.042	.194	.241	.181	.439
100	.199	.306	.300	***.569	.232	.276	.198	.264	.082	.245
200	.286	.386	.300	.505	**.366**	**.562	.209	***.407	.220	.363
500	.302	.389	.285	.315	.325	.396	.185	.315	.146	.244

Table 2.10: LSC results.

	SUBJ		DIR-OBJ		PP-OBJ		*all*	
50 training iterations								
20	**.189**	.503	.209	**.509	**.062**	**.045**	.121	.377
50	.094	.208	**.258**	.360	**.062**	**.045**	.070	.444
100	.094	.208	.041	.074	**.062**	**.045**	.134	.400
200	.094	.208	.041	.074	**.062**	**.045**	.060	.367
500	.094	.208	.041	.074	**.062**	**.045**	.094	.405
100 training iterations								
20	.185	**.507	.229	.495	**.062**	**.045**	.114	.429
50	.094	.208	.222	.478	**.062**	**.045**	.107	***.465
100	.094	.208	.041	.074	**.062**	**.045**	**.141**	.385
200	.094	.208	.041	.074	**.062**	**.045**	.081	.442
500	.094	.208	.041	.074	**.062**	**.045**	.099	.343

Table 2.11: PAC (EPS-a) results.

	SUBJ		DIR-OBJ		PP-OBJ		*all*	
50 training iterations								
20	.213	.570	.150	.502	.065	.356	.070	.358
50	.214	.513	.160	.719	.113	.243	.099	.425
100	.244	.463	.047	.626	.222	.401	.085	***.543
200	.255	.510	.258	.676	.206	.347	.210	.432
500	**.272**	.597	.198	.755	.238	**.481	.174	.489
100 training iterations								
20	.201	***.651	.160	.450	.065	.247	.083	.345
50	.237	.513	.172	.709	.113	.098	.105	.373
100	.222	.443	.165	.692	**.288**	.379	.095	.439
200	.222	.090	.265	.498	.179	.359	**.217**	.142
500	.233	.497	**.305**	***.795	.257	.323	.189	.249

Table 2.12: PAC (EPS-b) results.

2.3.2 Verb Sense Disambiguation

Introduction

Word sense disambiguation has a long history (see Agirre and Edmonds (2006) for an overview) but still remains a core problem to many NLP applications such as message understanding, machine translation, and question answering. Especially the disambiguation of highly polysemous verbs with subtle meaning distinctions is difficult. The definition of sense inventories is also challenging, controversial, and not equally appropriate across NLP domains (Ide and Wilks, 2006).

In the following, we describe experiments on Verb Sense Disambiguation (VSD). The VSD relies on the EPS model whose clusters are interpreted as 'sense labels'. As these labels are unlikely to exactly match the senses of some independently defined sense inventory, the cluster labels must be mapped to the existing senses in order to use the clustering model for their disambiguation. The mapping is done by a statistical classifier which is trained on manually sense tagged text. The classifier computes the probability of each possible verb sense given the cluster labels.

As introduced earlier, the verb clustering model is based on the assumption that verbs which agree on their selectional preferences belong to a common semantic class. For example, the two verbs *sit* and *lie* in Example 2.9 belong to a class of verbs which describe an entity placed on top of another entity.

(2.9) *The cat sits/lies on the sofa.*

Different readings of a verb usually differ in argument preferences. Example 2.10 shows two readings of the verb *roll* with different subcategorisation frames.

(2.10) *The thunder rolls. – Peter rolls the ton off the road.*

Example 2.11 demonstrates that also the class of arguments (weaponry vs. employee) can differentiate between verb meanings.

(2.11) *fire a gun – fire a manager*

These differences in subcategorisation and selectional preferences allow the clustering model to assign the readings of a verb to different clusters, which can then be used as evidence for verb sense disambiguation. We implemented a VSD system based on these ideas and evaluated it on Senseval-2[8] data.

[8] http://193.133.140.102/senseval2/, last visited June 2009

The Senseval-2 Data

The Senseval-2 shared task was a word sense disambiguation (WSD) competition for nouns, verbs and adjectives. In our experiments, only the disambiguation of verbs is considered. We tested our system on the English Lexical Sample task of the Senseval-2 data set, which contains 3,565 verb instances in the training set and 1,806 in the test set. These data comprise 29 different target verbs with 16.76 senses on average. This high polysemy rate is due to the fact that particle verb constructions such as *carry on* are considered senses of the base verb. Particle verbs are explicitly marked in the corpus, which facilitated disambiguation because it allowed the elimination of inappropriate readings. The Senseval-2 data are hand-tagged with one (sometimes two) WordNet sense keys of the pre-release WordNet version 1.7. The inter-tagger agreement (ITA) of the task was only 71.3% which can be taken as an upper bound for this task.

Preprocessing of the Data We parsed the Reuters corpus with the BitPar parser (Schmid, 2006) and extracted the verbs and their arguments. With the extracted tuples (cf. Section 2.2.1), we trained the verb clustering models.

The Senseval-2 corpus was also parsed with the BitPar parser but only with respect to the verbs to be disambiguated and their arguments. For each tuple, we calculated the cluster probabilities according to the verb clustering model, cf. Equation 2.12. Cluster probabilities below a threshold of 0.1 were ignored.

$$p(c|tuple) = \frac{p(c, tuple)}{\sum_{c'} p(c', tuple)} \tag{2.12}$$

Training of the Classifier Next we used the Senseval-2 training set to train a classifier that estimated the probability of senses within a cluster. If c is a cluster and s is a sense, we first summed over the probabilities of c for any tuple that was labeled with s. This gave us the frequency of the joint occurrence of s and c.

$$f(s, c) = \sum_{tuple:sense(tuple)=s} p(c|tuple) \tag{2.13}$$

To get the probability of s given c, we calculated the relative frequency. The probabilities of all different senses within c therefore sum up to 1.

$$p(s|c) = \frac{f(s, c)}{\sum_{s} f(s, c)} \tag{2.14}$$

Sense Classification The classifier assigns a sense to each tuple based on the verb, the cluster probabilities, and the sense probabilities. The most probable clusters of a tuple are obtained from the clustering model, and the sense probabilities for these clusters were estimated in the training. The classifier multiplies the probability of each cluster with the probability of each

sense of the cluster. The total probability of a sense for a given tuple is computed by summing over all clusters:

$$p(s|tuple) = \sum_c p(c|tuple)p(s|c) \tag{2.15}$$

For example, assume that the cluster probabilities for the verb-argument tuple
$\langle carry, subj{:}obj, man, suitcase\rangle$
are c1=0.94, c2=0.05. The classifier would provide for c1: sense1=0.18 and sense2=0.81, whereas c2 would hold sense1=1 as a single sense. In this case, the most probable sense would be $p(s_2|tuple) = p(c_1|tuple)p(s_2|c_1) + p(c_2|tuple)p(s_2|c_2) = 0.94 * 0.81 + 0.05 * 0 = 0.76$.

In accordance with the Senseval scoring we counted each verb with an identical sense tag as a match (Kilgarriff, 2000). If no sense was found,[9] the most frequent sense (MFS) of the verb was assigned. If no MFS existed because the verb was not in the training data, we randomly chose one of the senses of the verb in WordNet1.7 and took 1 divided by the number of senses as the estimated correctness of this random decision.

Evaluation

The system was optimised on the training set of the English Lexical Sample task. All experiments that follow in this section are done on this data set with a tenfold cross-evaluation. We experimented with different settings of the model and the preprocessing to find the best features.

We established a base system to explore the performance of our features. The base system uses a PAC clustering model with 50 clusters, and 100 training iterations. In addition, we compared the results to the MFS baseline which assigns all verbs to their most frequent sense.

If nouns from the verb-argument tuple were not in WordNet, we replaced them by a placeholder $\langle UNKNOWN\rangle$. Additionally we used the placeholder $\langle NONE\rangle$ when the parser failed to find the head of an argument (e.g. the subject in subject-less sentences).

For significance testing, we applied a Binomial test and considered only tuples that where classified correctly either in the base system or in the experiment system but not in both. We chose an significance threshold of 5%.

Experiments on the Data Since the variable frame size and the conceptualisation of the arguments were an extension from LSC to PAC we aimed to discover to what extend the frames and arguments helped in the classification process. We tried to gradually increase the amount of information provided by the arguments. First we replaced the arguments in the Senseval2 and the Reuters tuples by the placeholder 'x' to use only information given from the frames. A tuple extracted from the sentence *"He began a battle"* is represented as $\langle begin, subj{:}obj, x, x\rangle$.

[9] This can be due to parsing errors or because the assigned clusters did not appear in the training data with that verb

In a second experiment we eliminated the generalisation to concepts in PAC. This means that the probability $p(a_i|c, f, i)$ in Equation 2.1 is directly estimated from data. A mapping of WordNet-unknown words is not required here. The above tuple would look as follows: $\langle begin, subj{:}obj, he, battle\rangle$

In a third experiment, we replaced pronouns that are likely to refer to humans such as *I*, *he*, *us* etc. by the WordNet concept 'person'. Other arguments not covered by WordNet were again mapped to $\langle UNKNOWN\rangle$. Our example tuple turns into: $\langle begin, subj{:}obj, person, battle\rangle$.

Table 2.13 shows that the difference between no arguments at all and the base system amounts to only 2%. That means that the classification is mostly done by the subcategorisation frame. Selectional preferences improved performance just slightly. The data set where pronouns were mapped to 'person' shows the best results.

In the version without WordNet the arguments caused more damage than they helped. This was a problem of data sparseness. A given tuple with an argument a could only be assigned to a cluster if the model contained a in the same cluster, the same frame and the same slot. Because the corpus was not large enough it happened quite often that a tuple with a rare frame did not fit into any cluster. For comparison: in our 'no wordnet' data set 107 tuples out of 356 did not belong to any cluster. In the base system this happened only 21 times. This means that if we use detailed information about frames we have to generalise the nouns or we need much more data.

no arguments	53.40
no wordnet generalisation	50.23
base system	55.68
pronouns to 'person'	56.88

Table 2.13: Manipulating the arguments.

Experiments on the Model

1. Number of Clusters In this experiment we trained clustering models with different numbers of clusters (see table 2.14)[10]. If the number of clusters was rather small, more senses were united in one cluster causing mis-classifications. An inspection of the cluster parameters of a model with 20 and 160 clusters[11] for the verb *begin* showed that the average number of *begin*-senses in the 20 cluster model was 4.0 senses per cluster, where 13 clusters contained the verb *begin*. The 160 cluster model had 72 clusters that contained this verb with an average number of *begin*-senses of 2.72. The total ambiguity rate of the verb *begin* was 8.

[10]Significance testing yielded values over 0.05%. Values that got close to the threshold are nominated.
[11]Only clusters with a probability over 0.01 were considered.

c 20	54.72 (significance: 0.07)
c 40	55.90
c 50 (base system)	55.68
c 60	55.28
c 80	55.52 (significance: 0.05)
c 100	55.85
c 120	56.01
c 140	56.04
c 160	56.58 (significance: 0.07)
c 180	56.69 (significance: 0.05)
c 200	55.96

Table 2.14: Variation in the number of clusters.

Although the results were not significant, a tendency towards an improvement at higher cluster numbers was visible. It seems that the more clusters we defined the more consistent the clusters were and the better the sense classification turned out. If the number of clusters is too high, we would expect a data sparseness problem because the number of tuples per cluster decreases and the probability estimates become unreliable. Maybe this point is reached with 200 clusters.

2. Number of Iterations It was often observed that the performance of systems which are trained with the EM algorithm improves over a couple of iterations and then starts to decrease again. Our experiments on the number of iterations show that further training iterations did not make a significant difference after the 30th iteration (see table 2.15[12]). After 30 iterations the results bounced up and down randomly. However, even after 100 iterations we did not reach a turning point where results got noticeably worse.

Comparing LSC and PAC Since the LSC model does not include the frame in its parameters and since the number of arguments must be fixed, we used a different tuple representation for LSC. We created a pseudo argument containing the frame and we chose only subject and object arguments (which are undefined if not contained in the frame): ⟨begin, subj:obj:p:np, it, visit⟩

If we applied LSC to a data set without arguments, the result was similar to the corresponding PAC result (see table 2.16). If we added arguments as described above, we got 50.65%. In this experiment the model was losing out because it was trained on a rather small data set[13] and had similar data sparseness problems as the PAC version without WordNet. If we used a larger

[12]Values marked with an asterisk are significant results compared to the base system.

[13]The small data set contains only tuples with words existent in WordNet (2.4 million Tuple).

	c 20	c 50	c 100	c 180
i 10	51.05*	52.45*	53.38*	53.63*
i 20	54.25*	54.05*	55.06	55.06
i 30	54.50*	55.25	55.62	55.09
i 40	54.13*	55.82	55.59	56.24
i 50	54.19*	55.79	55.51	55.76
i 60	54.05*	55.68	55.42	56.01
i 70	54.38*	55.95	55.68	56.32
i 80	54.55*	55.65	55.93	56.60
i 90	54.41*	55.70	55.59	56.80*
i 100	54.72	55.68	55.85	56.69

Table 2.15: Variation of the number of iterations.

training set[14], performance improved considerably (see the last row of table 2.16). The result shows that LSC suffers more from data sparseness than PAC which indicates that the argument generalisation helps.

	LSC	PAC
no arguments	53.07	53.40
arguments, small corpus	50.65	55.68 (base system)
arguments, large corpus	55.03	56.45

Table 2.16: Comparing LSC and PAC.

Results

The final evaluation was carried out on the test data of the English Lexical Sample task with the best combination of features according to the previous experiments. That was the data set where the pronouns were partially mapped to the WordNet concept 'person'. The model was trained on a large data set with 180 clusters and 90 iterations. Table 2.17 compares our results to the accuracy scores of other WSD systems on this task for verbs[15].

The performance of our system is close to that of the best system in the Senseval-2 evaluation (Seo et al., 2001) but somewhat behind current state of the art (Chen and Palmer, 2009). However, it must be pointed out that we used very few features – only subcategorisation frames and

[14]In the large data set all tuples provided from the Reuters corpus were taken. Words not included in WordNet were replaced by a placeholder (4.9 million tuple).

[15]Listings of the English Lexical Sample results of verbs can be found in Dang and Palmer (2002)

MFS	46.1
Seo/Lee	57.6
Dang/Palmer	59.6
Chen/Palmer	64.6
PAC	57.06

Table 2.17: Results on the evaluation data set.

arguments provided from the clustering model–, and that our results are likely to improve with additional features. Seo et al. (2001)[16] used no linguistic information at all, but took into account local contexts, topical contexts and bigram contexts. These features seem to be quite different from ours. Incorporating them into our system would probably improve the performance.

Error Analysis and Future Work We had to deal with errors on different levels. Besides of parser errors – in the Senseval-2 training set 4.1% of the target verbs were not returned – we had the problem that the information in the tuples was often incomplete. Our Senseval-2 data set contained in 2,669 out of 3,565 tuples one or more placeholders corresponding to arguments missing in WordNet or to unrecognised objects. If we mapped pronouns that referred to humans to the concept 'person', still 2,169 tuples contained a placeholder, but results got better. This indicates that future work should concentrate on data preprocessing with anaphora resolution and named entity tagging.

To avoid the bottleneck of manually annotated training data, we would like to turn our supervised system into an unsupervised system by taking the ID of the most probable cluster as the 'verb sense'. To get an intuition of how well our system covers the senses with the clusters, we chose the most frequent clusters for the verb *begin* in a 160-cluster model and looked up the most probable senses included in these clusters. In the following, clusters and senses are listed in descending order according to the frequency or probability respectively. The verb *begin* has eight senses in the Senseval-2 data. The MFS begin%2:30:00:: was covered in several clusters (c110, c14, c21, c26, c128), which all selected for the frame $subj$:s[17]. It was interesting to see that the clusters listed above chose different arguments. c110 selected for a location as a subject, where as c14 selected for a process, c21 for a physical object – which seems to be a very general cluster – c26 for a person and c128 for an abstraction. This means that this model fractions the sense into finer grained sense distinctions than WordNet does. The sense begin%2:42:04:: was included in c119 and c75 both holding the intransitive frame and again selecting for different argument concepts: 'process' and 'person'. The sense begin%2:30:01:: is modeled about as well as the described ones.

[16] http://www.informatics.susx.ac.uk/research/groups/nlp/mccarthy/SEVALsystems.html#kunlp, last visited June 2009

[17] 's' is a sentence slot.

It was more difficult to model the sense begin%2:42:00:: which occurs only 24 times out of 508 *begin*-instances. Besides its sparseness it is very similar to sense begin%2:42:04::. The WordNet description for the former is: 'have a beginning, of a temporal event' and for the latter: 'have a beginning, in a temporal, spatial, or evaluative sense'.

Sense begin%2:42:03:: shows that our system has problems if a sense occurs with different subcategorisation frames. This sense was only tagged correctly if it occurred with the frame $subj$:p:np. It must be pointed out though that we had only 17 instances of this sense in the Senseval-2 corpus. The remaining three senses were never chosen by the system because they occurred very rarely (seven times or less).

Verb Sense Disambiguation with Named Entity Recognition

In the shown experiments the selectional preferences did not improve results as much as we expected. That is why we had a closer look at the data. Table 2.18 gives some examples of Senseval-2 tuples, where the first column specifies the sense, the second the subject, and the last one the object of the highly ambiguous verb *carry*. It shows that the nouns selected by the verb, group well on a higher abstraction level.

carry	subject	object
42:01	Mr. Baker (person)	weapon (artifact)
42:01	he (person)	glass (artifact)
42:02	dept (abstract)	guarantee (abstract)
42:02	bill (abstract)	ban (abstract)
42:12	woman (person)	significance (abstract)
42:12	man (person)	stigma (abstract)
42:03	plane (artifact)	bomb (instrumentality)
42:03	she (= a ship) (artifact)	rigging (instrumentality)

Table 2.18: Selectional Preferences for *carry*.

These examples encouraged us to perform more experiments where we tried to collect more detailed information for the arguments. The experiment settings were the same as explained above. Changes were done in the pac-train version – we used the improved version pac-train-2.1 implementing ESP-b in contrast to the earlier version implementing ESP-a – and in the preparation of the data. We used the Stanford Named Entity Tagger[18] for each target sentence of the Senseval-2 data set and for the Reuters corpus and extracted the verb-argument tuples. In the resulting tuple set (NER data set), entities labelled as 'person' were replaced by the pseudo-word '<PERSON>', organisations were replaced with '<ORGANISATION>', and locations

[18]http://nlp.stanford.edu/software/CRF-NER.shtml, last visited 9/25/2010

turned to '<LOCATION>'. In addition, all pronouns referring to humans were also replaced with the pseudo-word <PERSON>. In comparison, we used a data set where only pronouns that referred to humans were replaced with the pseudo-word <PERSON> (PRON data set), but named entities were left untouched, similar to the above described experiments.

We trained 50 cluster models with the modified Reuters corpus[19]. Before training, we had to adopt our WordNet (WN) version to the new data set. We inserted the pseudo-words into the according WN synsets: ⟨PERSON⟩ is inserted into synset 17 (*person, individual, someone, somebody, mortal, soul*), ⟨ORGANISATION⟩ is located in synset 44511 (*administration, governance, organisation* amongst others) and ⟨LOCATION⟩ occurs in synset 35 with the synonym *location*.

The experiments were carried out on the training set of the Senseval-2 English Lexical Sample task with a tenfold cross-evaluation. The results, however, turned out to be worse with Named Entity Recognition (NER) as shown in Table 2.19. The difference between the systems had a significance value of 0.07.

	PRON data	NER data	MFS
c50	54.84	53.69	46.42

Table 2.19: Comparing NER system with PRON system.

In another experiment, we modified our WN file. Instead of integrating the pseudo-words into synsets, we created new synsets for each pseudo-word and placed the synset rather high in the WN hierarchy. The results did not change a lot (see Table 2.20).

	NER data
pseudo-words deep in WN tree	53.93
pseudo-words high in WN tree	53.82

Table 2.20: Comparing different WN versions (models were trained in 80 iterations).

To find reasons for this result is difficult. On the one hand, results get clearly better because of the replacement of pronouns. This suggests that further improvements in the argument extraction process would help. On the other hand performance gets significantly worse when in addition to the pronouns, the named entities are replaced. Possibly the named entities provided by the NER tool are qualitatively not correct enough to help in the VSD task. The NER process inserts some systematic errors. For example, the tuple: ⟨begin, subj:obj, Europe, crusade⟩ of the PRON data set correspond to the tuple ⟨begin, subj:obj, ⟨LOCATION⟩, crusade⟩ of the NER data set. Though in this context *Europe* is rather an organisation than a location.

[19]The models are very large, because hardly any filtering of the training tuples was done. However, the models return better results with smaller, less noisy training data.

PAC-1.0 vs. PAC-2.1 for VSD

In these experiments we wanted to find out whether the new PAC-2.1 Model (ESP-b) outperforms the PAC-1.0 Model (ESP-a) in the Senseval-2 task. Our hypothesis was that results would be better, because of better modeling of the arguments.

We used the PRON data set as described in 2.3.2 which is trained with PAC-2.1 and trained an equivalent model with PAC-1.0. For significance testing, we used the same method as in the former VSD-experiments: a Binomial test with a significance threshold of 5%. As shown in table 2.21 our expectations came true with a significance value of 0.35.

	PAC-1.0	PAC-2.1	MFS
c50	54.39	54.95	46.42

Table 2.21: Comparing PAC-1.0 and PAC-2.1.

To get an intuition of what had changed, we had a closer look at some *begin*-tuples, that were annotated according to the gold standard (GS) using the new system but had failed previously. Almost all corrected tuples of the sense *begin%2:30:00::* were formerly annotated with sense *begin%2:42:04::*, like for example the intransitive tuple: $\langle begin, subj, life\rangle$ (GS: begin%2:30:00::). In addition, we selected a very similar example tuple, with the GS-sense *begin%2:42:04::*: $\langle begin, subj, strike\rangle$. The latter was labelled correctly in both systems. The two example tuples differ only in their argument selection. But even the arguments refer both to an abstraction in terms of the WordNet hierarchy.

Table 2.22 shows the clusters that were selected by the PAC model, and the classification result of the two example tuples. The very dominant subcat frame of the *begin%2:30:00::*-sense is *SUBJ:S* where the most frequent frame of *begin%2:42:04::* is the intransitive frame. This is probably the reason, why the PAC-1.0 model selected the same cluster for both examples and therefore chose the same sense, which was incorrect. In contrast, the PAC-2.1 model was able to distinguish that the concept *strike* was an event and *life* was a *state* even though both were abstract entities. Probably it was this fine difference that caused the selection of the GS sense. This might be a sign that the subcat frame in the new model was less dominant than in the old system (see 2.3.2) and that the arguments gained more importance in the classification process.

2.3.3 Semi-Supervised Sense Labelling for German

Introduction

We present a semi-supervized verb sense disambiguation technique for German data which is based on PAC.

	PAC-1.0		PAC-2.1	
argument	strike	life	strike	life
sense labelling	begin%2:42:04::	begin%2:42:04::	begin%2:42:04::	begin%2:30:00::
cluster	c17 (0.99)	c 17 (0.99)	c30 (0.54) c48 (0.38)	c16 (0.32) c41 (0.30) c48 (0.22) c30 (0.15)

Table 2.22: Cluster selection of two example tuples in PAC-1.0 and PAC-2.1 models

The application we have in mind is a text mining task as performed by SemTrack[20] where communication verbs – in a wider sense – need to be retrieved. So far, the project uses a list of communication verbs for their analyses. This list is the result of a theoretical linguistic project from the Institut für Deutsche Sprache Mannheim (Harras et al., 2004), which is very detailed and therefore very ambiguous.

The goal is, to label a verb in context as either a communication verb or a non-communication verb. For example the verb *anführen/lead, cite* in the context *Truppen anführen/lead troups* would not be of interest, whereas *ein Beispiel anführen/cite as an example* would be considered as a communication verb.

Method

We tried to influence the clustering algorithm such that it forces a labelled tuple into a predefined cluster and then group similar tuples with the labeled tuple. For example: the tuple: <erzählen, SUBJ:OBJ, Lehrer, Geschichte> (<tell, SUBJ:OBJ, teacher, story>) is labelled with the sense *Telling* and would be forced to be in cluster 1. During training the labeled tuple stays in cluster 1 with a high probability, and is joined by similar tuples. After the training, cluster 1 can be considered as a '*Telling*'-cluster and the verbs within the cluster as communication or '*Telling*' verbs.

Data

So far, there is very little work done on the field of WSD for German. That is why hardly any annotated data is available. The only resource known to the authors is the Salsa corpus[21], where verbs (and other predicates) are labelled with frames in the sense of FrameNet. In this work, the Salsa frames are taken as word senses.

[20] laboratory for computer based meaning research, http://www.semtracks.org/web/, last visited 09/25/2010

[21] http://www.coli.uni-saarland.de/projects/salsa/page.php?id=index, last visited 09/25/2010

There are quite a few Salsa frames that could be subsumed in a wider sense as communication frames: *Telling, Communication, Communication_response, Communication_manner* to name a few. However, except of *Telling* all of these senses occur less than 20 times in the corpus. That is why we decided to only use tuples with the sense *Telling* as targets for a start, since they are very frequent in the corpus (1,479 Targets).

The Salsa corpus is used as a source for manually annotated data. In addition, the HGC corpus is used to obtain enough training data for PAC. Both corpora are parsed by BitPar, and the tuples are automatically extracted. We could have used the manually annotated syntactic information of the Salsa corpus. Thus to make sure, that the tuples of both data sets are equivalent, we extracted the Salsa tuples just like the HGC tuples, by means of the parser.

The Salsa corpus is randomly split into a training set (9,460 tuples) and an evaluation set (3,153 tuples with 372 *Telling* tuples). The training set was merged as labelled tuples with the HGC tuples. The labelled target-tuples in the training set are marked with a cluster number which after training will be the cluster with *Telling* verbs, e.g. <erzählen, SUBJ:OBJ, Lehrer, Geschichte, 1> (<tell, SUBJ:OBJ, teacher, story, 1>).

Sense Classification and Evaluation

For the classification of the verbs, we trained a classifier in the same manner as in the former VSD experiments using the Salsa training data. If c is a cluster and s is a sense, we first summed over the probabilities of c for any tuple that was labeled with s. This gave us the frequency of the joint occurrence of s and c.

$$f(s,c) = \sum_{tuple:sense(tuple)=s} p(c|tuple) \tag{2.16}$$

To get the probability of s given c, we calculated the relative frequency. The probabilities of all different senses within c therefore sum up to 1.

$$p(s|c) = \frac{f(s,c)}{\sum_s f(s,c)} \tag{2.17}$$

The only difference to the former approach is, that in the former experiments each verb is treated separately in the training process, i.e. the probability $p(s|c)$ is calculated for each verb and not for all verbs in total.

Sense classification is accomplished exactly like in the Senseval-2 experiments:

$$p(s|tuple) = \sum_c p(c|tuple)p(s|c) \tag{2.18}$$

For evaluating the results, we calculated the f-score and compared our results with the most frequent sense (MFS) baseline. If a tuple could not be classified, it was considered to be a non-*Telling* tuple.

Experiments and Results

We accomplished experiments with several models and data sets. We varied the amount of slots and fillers within the tuples. The standard data set contained no modifier-PPs. In another data set the tuples contained all PPs that were extracted by BitPar. Finally we created a data set where all PPs within a tuple were eliminated.

We chose different granularity levels in terms of the classification of the labelled data. We trained models where all labelled tuples were forced into one cluster (coarse grained model).

In other models, the *Telling* tuples were allocated to several clusters according to the subcat frame. All *Telling* tuples that shared the same subcat frame were grouped into one cluster (fine grained model).

The results are listed in Table 2.23 and 2.24. Unfortunately, the MFS baseline is unbeatable high

model	precision	recall	f-score
c 30	0.32	0.97	0.48
c 50	0.32	0.94	0.48
c 100	0.36	0.94	0.53
c 150	0.38	0.94	0.54

Table 2.23: Variation in the number of clusters in a fine-grained model.

(0.97 f-score), such that the results did not even come close to the baseline (see Table 2.24).

model	precision	recall	f-score
fine grained	0.32	0.94	0.48
coarse grained	0.53	0.98	0.68
all PPs considered, fine grained	0.41	0.91	0.56
no PPs considered, fine grained	0.32	0.97	0.47
unsupervised model	0.49	0.98	0.65
MFS baseline	0.94	0.98	0.97

Table 2.24: Different 50 cluster models in comparison.

Still the experiments show the following.

- The semi-supervised model outperforms the purely unsupervised model.
- Higher numbers of clusters seem to be better, which confirms the results of the Senseval-experiments.

- Performance of the coarse grained model are more accurate than fine grained models. Probably the fine grained models are far too differentiated to be successful.
- Detailed PP information seems to be helpful. Here, the fine grained model is quite successful.

Cluster Analysis and Discussion The *Telling* cluster, generated by the coarse grained c50 model, looks quite encouraging. The cluster is the most probable cluster of the model and contains verbs and subcat frames as shown in Table 2.25. The last column gives information if the verb occurred as a labelled verb in the training corpus. Quite a few communication verbs

Cluster1	0.089902	labelled
verbs		
sagen	0.251515	+
erklären	0.092334	+
meinen	0.064200	-
berichten	0.050434	+
mitteilen	0.048037	+
betonen	0.030940	-
wissen	0.021287	-
weißen	0.014975 (probably a lemmatisation error)	-
schreiben	0.013983	-
ankündigen	0.012152	-
erzählen	0.011936	-
glauben	0.011220	-
fordern	0.010654	-
bestätigen	0.010318	-
feststellen	0.010299	-
subcat frames		
SUBJ:S-OC (12)	0.674082	
SUBJ (1)	0.202031	
SUBJ:OBJ (2)	0.071516	

Table 2.25: 'Telling'-cluster of a coarse grained c50 model.

group to the labelled verbs due to the clustering process like *meinen/mean, betonen/emphasise, erzählen/tell* and others. Non-communication verbs like *wissen/know* probably occur in the cluster, because of the very prominent subcat frame *SUBJ:S-OC*, that fits *wissen/know* as well. It is interesting that the subcat frame with Dative (in: *Er sagt ihm .../He told him ...*) did not occur in the model with a probability above the standard threshold. This indicates that a more fine-grained distinction of the *Telling* tuples would be helpful. Possibly performance gets better, if

two clusters are selected as *Telling* clusters: one cluster for tuples with the very frequent slot *S-OC* (*dass/that* sentence) and a second cluster for tuples with a Dative slot.

Mainly the precision values turned out to be problematic, because quite a few non-*Telling* verbs were allocated in the *Telling* cluster. These were verbs with rather low probabilities. Better results were achieved when only verbs with a probability of at least 0.25 where taken (0.71 f-score with the coarse grained c50 model). Another way to improve precision could be a post-processing step, with a list of (ambiguous) communication verbs like the above mentioned list of (Harras et al., 2004). Non-communication verbs with very similar selectional preferences like *wissen/know* and *glauben/believe* could be this how excluded from the *Telling* cluster.

So far the system is in the early stages of development. Many parameters like number of clusters, probability threshold, granularity level etc. could be optimised. However, the main problem of the system is the very high MFS baseline for our example sense *Telling*. The *Telling* verbs of the Salsa corpus are (i) not very ambiguous and (ii) *Telling* verbs that have another MFS like *bedeuten/mean* hardly occurred.

Besides the problem that communication verbs are very diverse and unspecific in terms of selectional preferences, the Salsa data don't provide optimal information for the task. The manual labelling of the data is done verb by verb and not sentence by sentence which means that seldom verbs tend to have no annotation yet. This approach fits quite well to a classical WSD task where certain verbs are to be disambiguated. Yet, when verbs with a certain sense need to be retrieved, it is important to have the rather inconsiderable verbs that rarely occur in text as well, besides the very common ones. Otherwise no realistic distribution over all communication and non-communication verbs is given. These rather uncommon verbs are the examples which need to be generalised, thus these are the examples where clustering could improve results.

Future Work It would be interesting to see, if results get better when using different seeds for the model training. One option might be to get training material from Harras et al. (2004) instead of using the Salsa data. Harras et al. (2004) make extensive use of examples taken from written German corpora. These examples would provide enough material for training. In addition, all communication verbs would be covered.

Chapter 3

Implementation

Section 3.1 describes the implementation of the *node pruning strategy (NPS)* model, and Section 3.2 describes the implementation of the *edge pruning strategy (EPS)* model.

3.1 Model 1 (NPS): *LSCpref*

The NPS model is called *LSCpref*, as it represents an extension of the *LSC* approach (Rooth et al., 1999).

3.1.1 Overview

The stochastic process underlying the probabilistic model in Section 2.1.1 on page 3 is implemented as a graph that captures all possible ways of analysing or generating a particular tuple $\langle v, f, a_1, ..., a_{n_f} \rangle$. The graph is implemented as a directed acyclic graph. Transitions of the stochastic process are modelled by edges which are annotated with the transition probabilities. States of the stochastic process are modelled by nodes. Nodes are either of the type `AND`-node or `OR`-node. `AND`-nodes force the stochastic process to follow *all* edges from the node with probability 1.0. In `OR`-nodes the stochastic process follows exactly one edge from that node with the annotated probability. The graph directly corresponds to the PCFG defined in Section 2.1.2.

NB: The principal aim of our implementation is to *analyse* tuples $\langle v, f, a_1, ..., a_{n_f} \rangle$ by computing a probability, an inside probability to be precise, for them. Therefore, we decided to build the directed graph by starting with the elements of the tuple and ending in a unique state (*viz* TOP-node or TOP-state) which represents the whole tuple. The shape of the graph is thus a reversed tree; the TOP state corresponds to the start symbol of the PCFG, the start states correspond to the terminal symbols of the PCFG.

For every PCFG rule $A \rightarrow B_1...B_n$ associated with the probability p_A the following nodes and edges are added to the graph in the following way.

1. If the graph does not contain a node corresponding to the non-terminal A then add a new `OR`-node.
2. If $n = 1$ then
   (a) add a new `OR`-node for B_1,
   (b) add an edge from B_1 to A, and
   (c) associate that edge with p_A.
3. If $n > 1$ then
   (a) add a new unique `AND`-node that corresponds to $\langle B_1, ..., B_n \rangle$;
   (b) add an edge from $\langle B_1, ..., B_n \rangle$ to A, and
   (c) associate that edge with p_A;
   (d) add n new `OR`-nodes $B_1, ..., B_n$,
   (e) add an edge from each of these new nodes to $\langle B_1, ..., B_n \rangle$, and
   (f) add an associate each edge with the probability 1.0.

The principal data structures of the implementation are the classes `mdl_emgraph` and `subgraph`. `mdl_emgraph` implements the complete stochastic model which is to be trained. It consists of all relevant states, transitions and transition probabilities.

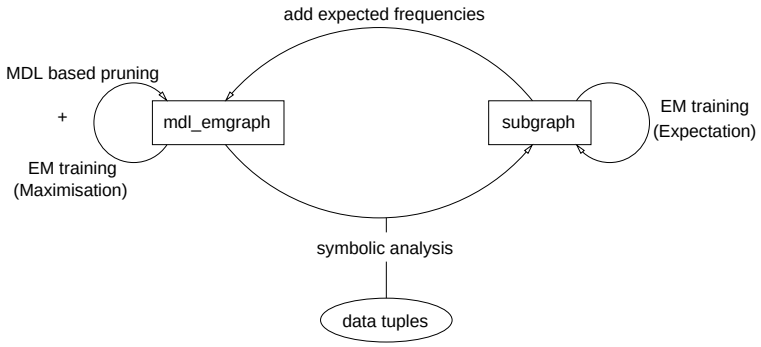

Figure 3.1: Overview of the MDL-EM training.

`subgraph` stores the part of the `mdl_emgraph` which provides part of the analysis of a certain data tuple. It comprises all states and transitions from the states corresponding to the elements of the tuple to the TOP-state. The iterative training process is briefly sketched in Figure 3.1. For each data tuple, the process *symbolic analysis* produces a `subgraph` consisting of all states and transitions of the `mdl_emgraph` that are visited with a probability greater than 0.0 when that tuple is analysed by the stochastic model. For each subgraph the *inside-outside* algorithm computes the expected frequency for each transition. These expected frequencies are added to the expected frequencies of the `mdl_emgraph`. After all tuples have been processed, a certain set of states and transitions in the `mdl_emgraph` is pruned according to some heuristics that are based on the MDL principle. After pruning, new probabilities are computed from the expected frequencies for all transitions of the `mdl_emgraph`. This step is called *maximisation*. Before

starting a new training iteration certain parts of the `mdl_emgraph` are expanded – these new parts are subject to pruning in the next iteration.

The implementation uses the following external resources:

1. WordNet which is a lexical resource from Princeton University and can be obtained from `http://wordnet.princeton.edu`;
2. the *Boost* graph library which is a free open source library that is developed at the site `http://www.boost.org`.
3. Online documentation of the files can be generated with the tool *doxygen*. It is called with the command `make doc` in the source directory. The files are generated in the relative directory *doc*.

3.1.2 Data Structures

Graph Implementation

The graph classes are based on the *Boost Graph package*[1]. The class hierarchy is shown in Figure 3.2.

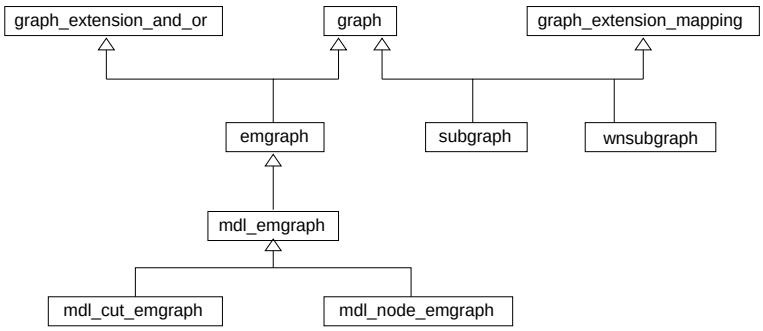

Figure 3.2: Class hierarchy of the graph classes.

The following classes are used in this implementation:

graph The graph class specifies a set of states and a set of transitions. Both entities are identified uniquely by integers between 0 and the number of states and transitions, respectively, in the graph minus one.

A graph object can be changed only by adding states and transitions, but note, that it is not possible to remove neither states nor transitions. States are added by the method

```
ensure_states(unsigned int).
```

Transitions are added by the method

```
connect(STATE, STATE).
```

If `connect()` is called with a state that is not a member of this graph, then it returns the constant

[1] See www.boost.org for detailed documentation.

no_state. There are no double connections between two states, that means, there are no two transitions in a graph with the same start and end states.

After changing a graph object the method compute_first_and_final_states() should be called. It analyses the whole graph and determines all start states of the graph, i.e. all states which don't have any incoming transitions, and all terminal states of the graph, i.e. all states which do not have any outgoing transitions.

This graph class supports the standard stream operators << and >>. Furthermore, it provides methods that output graphs in *GraphViz* format[2]. These methods can be parametrised with property maps or keymaps, see below, that hold properties of the states or transitions.

graph_extension_and_or This class is an extension to the graph class and it can only be used together with the graph class. It adds types to states, so that each state is either of type AND-state or OR-state. The default type is OR-state.

graph_extension_mapping This class is an extension to the graph class and it can only be used in combination with the graph class. This class adds the subgraph functionality. It provides mappings from the (*local*) states and transitions of this extended graph to the (*global*) states of another graph object.

Classes that are derived from this class have to override the following methods: connect(STATE, STATE), is_connected(STATE, STATE), ensure_states(unsigned int), get_num_states().

emgraph Graph for modelling the stochastic process that can be trained with the EM algorithm. States are of type AND-state or OR-state because emgraph is derived from graph_extension_and_or. Transitions are associated with the following two properties: (i) the transition probability encodes the probability that this transition is selected by the stochastic process given that the process is in the state where this transition starts; (ii) the transition frequency is used for EM training. It stores the total estimated frequency assigned by the training with all data tuple. This frequency is used in the maximisation step to compute a new transition probability.

The emgraph can be changed by adding states and transitions with the corresponding methods from the graph class.

Before starting EM training the emgraph has to be initialised, see for example initializeProbs_random(int, unsigned int) and initializeProbs_uniform().

EM training of this emgraph requires the classes subgraph and inside_outside. A precondition of the whole training process implemented here is that the emgraph has one unique topmost state (end state) and that there is a path from all other states in this emgraph to this topmost state.

mdl_emgraph This class is used as an abstract class which provides the functionality of pruning within the EM training algorithm. The pruning is guided by a heuristic that is based on miminimal description length (MDL).

This class specifically implements MDL-EM-training for hierarchies that are directed-acyclic-graph structures which have a unique topmost state.

The actual heuristics are defined in the subclasses mdl_node_emgraph and mdl_cut_emgraph

[2] see www.graphviz.org.

mdl_node_emgraph This class implements the abstract class `mdl_emgraph`. Pruning is node-wise. That means, for every leaf-node and the corresponding transitions in the selectional preference model the algorithm decides whether it should be pruned.

mdl_cut_emgraph This class implements the abstract class `mdl_emgraph` with a cut-wise pruning heuristics.

subgraph This class is derived from `graph` and `graph_extension_mapping` and implements subgraphs of emgraph objects. A `subgraph` is a graph which is associated with an `emgraph` object. And every state and every transition of this `subgraph` is associated with exactly one state or transition, respectively, from the associated `emgraph`. The transition probablities of this `subgraph` are directly taken from the corresponding transitions of the associated `emgraph`.

In EM training the `inside-outside` algorithm is applied to `subgraph`s.

Subgraphs are constructed by the procedures

- `emgraph::findNewSubgraph_special()`,
- `graph::add_to_subgraph()`, and
- `mdl_emgraph::add_upper_states_recursively`.

This `subgraph` class provide methods that output graphs in *GraphViz* format[3]. These methods can be parametrised with property maps or keymaps, see below, that hold properties of the states or transitions.

wnsubgraph This class is derived from `graph` and `graph_extension_mapping`. It is an auxiliary class that is used to filter the nodes and edges of a *WordNet* hierarchy which are relevant with respect to the given data.

Map Implementation

Properties of states and transitions can be efficiently handled by containers which are built upon vectors, because in every graph object all states and all transitions are identifiable by a continuous sequence of integers starting from 0 and ending at the number of states or transitions, respectively, minus 1.

These maps support the standard stream operators $<<$ and $>>$ to provide easy storing functionality.

property_map The `property_map` class template is a map implementation that is based on a vector. States and transitions directly correspond to the position in the underlying vector.

keymap The `keymap` class template is a the implementation of a bidirectional map or of an index. It assigns every object which is put into this `keymap` an integer number starting from 0. If this `keymap` already contains this object, then it simply returns the integer which it has already assigned to it.

Internally, properties are stored in a vector to model that every state or transition, respectively, is associated with a property, and properties are stored in a map, in order to model that every property is associated with exactly one state or transition, respectively.

[3] see `www.graphviz.org`.

Interface to WordNet

The class `wngraph` is the interface to the *WordNet* database using the princeton *WordNet* API in C. Every synset from WordNet and every noun from the lexicon is associated with a unique positive integer as a key.

- The method `getState(std::string)` returns the key of the lexical entry that matches the given string according to the WordNet API.
- The method `toString(unsigned int)` returns the string that is associated with the given key by the Wordnet API.

The actual `graph` object that stores the *WordNet* hypernym hierarchy can be retrieved with the method `getGraph()`.

The location of the database has to be specified by environment variables. It is either in `$WNHOME/dict` or in the directory specified by `$WNSEARCHDIR`. Since the WordNet API uses global state variables we apply the singleton-pattern which ensures that there is only one instance of this class for one process. A reference to the instance is retrieved with the static method `wngraph::getInstance()`.

3.1.3 EM Algorithm

The implementation of the expectation-maximisation (EM) algorithm has been briefly described on page 52 and Figure 3.1 illustrates the process schematically. The functionality of the EM training is implemented by the following classes:

emgraph see above.

subgraph see above.

inside_outside This class implements the inside/outside algorithm which computes the exptected frequencies for all transitions of a subgraph. Every `inside_outside` object is associated with a `subgraph`.

- The method `inside()` computes the inside probabilities of all states of the subgraph. A reference to the `property_map` with all inside probabilites is returned by the method `get_state_iprobs`.
  - If s is an `AND`-state with n incomming transitions from n lower states $l_1 \dots l_n$, then $inside(s)$ is computed as follows:

$$inside(s) = \prod_{i}^{n} inside(l_i)$$

– If s is an `OR`-state with n incomming transitions $t_1 \ldots t_n$ from the lower states $l_1 \ldots l_n$, then $inside(s)$ is computed as follows given that $p(t)$ is the transition probability of t:

$$inside(s) = \sum_{i}^{n} p(t_i) * inside(l_i)$$

- The method `outside(FREQ)` computes the outside probabilities of all states of the subgraph. Then it computes the expected frequencies for every transition and stores them in the `subgraph` with the method `subgraph::setTransFreq(TRANS,FREQ)`. A reference to the `property_map` with all outside probabilites is returned by the method `get_state_iprobs`.
  – Let l be a state and $t_1 \ldots t_n$ be the transitions from l to the upper states $s_1 \ldots s_n$:

$$outside(l) = \sum_{i}^{n} outside(t_n)$$

  where $outside(t_n)$ is defined as follows:
  – If s is an `AND`-state with n incomming transitions from n lower states $l_1 \ldots l_n$, then $outside(t_x)$ with $x \leq n$ is computed as follows:

$$outside(t_x) = outside(s) * \prod_{i \neq x}^{n} inside(l_i)$$

  – If s is an `OR`-state with n incomming transitions $t_1 \ldots t_n$ from the lower states $l_1 \ldots l_n$, then $outside(l_x)$ with $x \leq n$ is computed as follows given that $p(t)$ is the transition probability of t:

$$outside(t_x) = outside(s) * p(t_x)$$

em_analysis This class is intended to provide the methods that produce a `subgraph` of a given `mdl_emgraph` from a given data tuple.

So far this class does not exist. The functionality is spread over the classes `graph`, `emgraph`, and `mdl_emgraph`. The methods are

- `emgraph::find_subgraph_special(...)`,
- `graph::add_to_subgraph(...)`, and
- `mdl_emgraph::add_upper_states_recursively(...)`.

3.1.4 MDL Model

The MDL model is implemented by the class `mdl_emgraph` that manages all data structures and methods. It also integrates the specific MDL methods for both models as `virtual` methods.

The only property map that belongs to an `mdl_emgraph` is `my_state_item`, managing the number of nominal concepts that are below a certain state within the graph. This property is relevant for the mdl calculations, as we need to know how many nouns are below a certain WordNet synset. (The property map, however, manages this property for all states (and not only for WordNet states), as this is easier to

compute.) An `mdl_emgraph` is initiated by its number of states and transitions, and the above described property map.

In what follows, we provide a brief overview of the main classes and methods that are implemented for `mdl_emgraph`:

- putting and getting the number of lexical items in the graph (→ property map `my_state_item`);
- mappings between a priori and selectional preference WordNet states (in both directions, for states, transitions, etc.);
- checks on the activity of members and transitions within MDL models;
- auxiliary methods for MDL computations;
- calculation of selectional preferences (based on a comparison between probability paths in selectional preference models and the a priori model);
- printing routines for whole active graph; note that only "active" parts are printed (cf. details in the model descriptions below);
- eading routines for whole active graph: reads output of printing routines (and adds interpretation to induce all relevant data structures, cf. model descriptions below)
- classes for comparing/sorting probabilities/selprefs.

The class `mdl_cut_emgraph` is an `mdl_emgraph` for an MDL model that works cut-wise, cf. Section 2.1.3. It is implemented within three main steps, (1) the initialisation of the data structures, (2) an expansion of the current (i.e., from the last iteration) model towards its hyponyms, and (3) a pruning step. The pruning takes place after estimation but before maximisation, i.e., it works on frequencies as obtained in the very same iteration.

The model relies on three central data structures:

- a map of strings and sets of states `std::map<std::string, std::set<STATE>>` called `cfs_wn_map`: for each selectional preference model, there is a list of member states (with a priori model state numbers) that are part of the current cut;
- a map of strings and sets of states `std::map<std::string, std::set<STATE>>` called `cfs_graph_member` that are "active" members in the current selpref model, including the cut itself and hypernyms of the cut members up to the global wordnet top state; `cfs_wn_map` is a subset of `cfs_graph_member`;
- a map of strings and sets of transitions `std::map<std::string, std::set<STATE>>` called `cfs_graph_trans`: for each selectional preference model, there is a list of active transitions (with a priori model transition numbers), i.e. transitions that exist within the selectional preference model that are "active" and thus not pruned.

In what follows we provide an overview of the implementation of the three main steps within the MDL model.

(1) **mdl_init_wn_concepts**

initialises lists of WordNet concepts and transitions, for each selectional preference model: it uses the WordNet top state as only starting state for the cut members and all members (in `cfs_wn_map` and `cfs_graph_member`), and an empty list for the transitions (in `cfs_graph_trans`);

(2) **mdl_activate_new_wn_concepts**

expands the members in the current cut (unless they are terminal states) towards their hyponyms, for each selectional preference model; the expansion keeps `cfs_wn_map` as it was (i.e., in the later pruning step, the just expanded cut states are checked first); the member list `cfs_graph_member` and the transition list `cfs_graph_trans` are updated according to the hyponyms;

the transition probability between the new hyponym and its a priori concept is set to 1, and the transition probability between the cut member and its a priori concept is set to zero; the probability between the member and its hyponym within the selectional preference model is activated by the respective probability within the a priori model;

(3) **mdl_update_wn_concepts**

checks for each cut state and based on MDL calculations whether the expansion in the current iteration is kept or pruned back; if the hyponyms are pruned, the respective transition frequencies of in the selectional preference model are used to increase the frequency of the transition between the hypernym state in the a priori and the selectional preference model, and also to increase the frequency of the respective transition in the a priori model; the frequency transfer is propagated recursively downwards through the selectional preference model in the same way (as there might be hyponyms to the pruned hyponyms even if one of its transitions to a hypernym is pruned; member and transition lists are adjusted.

3.1.5 Tests

All above classes are tested with respect to their data structures and methods. I.e., for each class there is at least one `C++` test script that artifically sets up the data structured required by the specific class, and defines test cases that check on the various conditions of the class. The tests are used to ensure that –in case one has changed parts of the implementation– there was no change in the specific cases (and thus in the central meaning of the implementation). The tests are executed by `make testall`.

3.2 Model 2 (EPS): *PAC*

3.2.1 Data Structures

The main class of the PAC implementation is called **PAC**. A **PAC** object comprises

- a **WordNet** object for the noun hierarchy
- an array of **Cluster** objects for the model

- a back-off model and a prior model
- a set of tables (verbtable, frametable, slottable, argtable, frameslot) for mapping strings to indices and vice versa.

A **WordNet** object stores a noun hierarchy as a bidirectional graph by means of four arrays:

- **first_hyponym:** Position n of this array holds the position of the first hyponym of synset n in the array **hyponym**.
- **first_hypernym:** Position n of this array holds the position of the first hypernym of synset n in the array **hypernym**.
- **hyponym:** contains the sequence of hyponyms of all synsets starting with the hyponyms of synset 0, followed by the hyponyms of synset 1 and so on. The hyponyms of synset n are found at the positions ranging from first_hyponym[n] to first_hyponym[n+1]-1.
- **hypernym:** contains the sequence of hypernyms of all synsets. The hypernyms of synset n are found at the positions ranging from first_hyponym[n] to first_hyponym[n+1]-1.

A **Cluster** object comprises

- the cluster probability **prob**
- an object of class **SimpleSlotInfo** for the verb probabilities
- an array of **FrameInfo** objects for storing the selectional probability submodels.

A **FrameInfo** object comprises

- the index of the frame
- its probability
- an array of **SlotInfo** objects

A **SlotInfo** object stores the selectional preferences for one particular slot and comprises a **SimpleSlotInfo** and a **WNSlotInfo** object. Only one of the two subobjects is actually used. Which one is used is determined by the value of the variable **uses_wordnet**.

A **SimpleSlotInfo** object comprises a set of **IDProbFreq** objects. Each **IDProbFreq** object stores the index and the probability for one of the possible slot fillers. The submodel for the predicate probabilities is also represented with this data structure.

A **WNSlotInfo** object comprises a hash table which contains an entry for each WordNet link contained in the selectional preference model of the current slot. The has table maps the index of the WordNet link to an object of class **WNLinkInfo**. The index of a WordNet link is its position in the array **hyponym** (see above).

A **WNLinkInfo** object holds the probability of a link.

Each probability of a PAC model is associated with a variable called **freq** which is used to accumulate estimated frequencies from which the probability is later reestimated.

The other important data structure is the **EMGraph** class which implements the inside-outside algorithm. The data structure for the inside-outside computation is stored as an and-or-graph by means of five arrays:

- **first_expansion** Position n of this array contains the position in the array **first_subnode** where the first "or"-subnode of the current node is stored.
- **first_subnode** Each element of this array contains the position in the array **subnode** where the first "and"-subnode of the current "or"-subnode is stored.
- **subnode** Index of the respective child node

The child nodes of the first "analysis" of the node with index n are found at subnode[first_subnode[first_analysis[n]]] ... subnode[first_subnode[first_analysis[n]+1]-1]. There are six more arrays:

- **expansion_prob** holds the probabilities of the expansion of the "or"-nodes to the corresponding "and"-subnodes
- **expansion_freq** holds a pointer to frequency accumulator for the corresponding expansion probability
- **node_iprob** is used to store the inside probability of the ("and"-)nodes
- **expansion_iprob** stores the inside probabilities of the "or"-nodes
- **node_efreq** stores the estimated frequency of the nodes
- **expansion_efreq** stores the estimated frequency of the "or"-nodes

Finally there are two data structures for the representation of data tuples: **VATuple** comprises a verb index, a frame index, and an array of argument head indices. **FVATuple** additionally comprise a tuple frequency.

3.2.2 Training

The training program reads the training tuples one by one and creates an **FVATuple** object. An EMGraph object is built from the **FVATuple** object (**EMGraph::build**), and the inside-outside algorithm (**EMGraph::estimate_freqs**) is run to compute estimated frequencies for all links of the EM graph. The estimated frequencies are accumulated (**EMGraph::increment_freqs**) in the **freq** values of the PAC model.

After all training tuples have been processed, the model parameters are reestimated (**PAC::estimate_probs**) from the accumulated frequency estimates. The selectional restrictions models are pruned and then extended again (**PAC::extend_selpref**). Then the next training iteration starts. After a given number of training iterations has been completed, the resulting model is stored (**PAC::store**) in the output file.

EMGraph::build builds the top-level of the EMGraph with the cluster nodes and the predicate nodes, and then recursively calls **EMGraph::add_frame** to build the lower levels of the graph. **EMGraph::add_frame** builds nodes for the frame expansions and the expansion of "simple" slots (which do not use WordNet). **EMGraph::add_wngraph** is called to build the subgraphs forwith the WordNet synset expansions. **EMGraph::add_wngraph** in turn calls **EMGraph::add_hyponyms**.

EMGraph::estimate_freqs first calls **EMGraph::inside_probs** to compute the inside probabilities bottom-up and then computes the estimated frequencies top-down.

PAC::estimate_probs computes the cluster probability estimates, calls **Cluster::estimate_probs** to compute the frame probabilities, and **SimpleSlotInfo::estimate_probs** to compute the argument probabilities for "simple" argument slots whose selectional preferences are not generalised based on WordNet. **PAC::prune_slot** is called to prune the selectional restriction models of slots which do use WordNet, and to reestimate their parameters.

Chapter 4

User Manual

4.1 Model 1 (NPS): *LSCpref*

`LSCpref` outputs a probabilistic model (Section 4.1.1) that can be transformed into a reader-friendly representation by `params2cluster` (Section 4.1.2) and then visualised by a GUI called `ClusterViewer` (Section 4.1.3).

4.1.1 LSCpref

`LSCpref` is a command-line tool that expects

- a parameter file (as only parameter), and
- a WordNet for the language to be used, in a format that allows access to the information by the WordNet functions, cf. `http://wordnet.princeton.edu/doc`, Section 3 *(Library Functions)*.

The parameter file defines all parameters that are necessary to run `LSCpref`. The first three lines of the file are irrelevant (i.e., not read by `LSCpref`); the subsequent 11 lines match the following pattern:

```
<parameter name>TAB<parameter value>
```

The parameters are defined as follows.

- **`input_file`**: file with input training tuples

  Each input tuple must be organised according to a specific pattern:
  column 1: frequency of the tuple;
  column 2: verb;
  column 3: frame type (with the frame arguments separated by ":");
  column 4...n: lexical heads of the $n - 3$ arguments in the frame.

An example tuple with respect to this pattern is the following:
`100 drink subj:objD girl tea`

In addition to this format, the tuple definitions underlie further restrictions:

- The frame is not allowed to contain multiple identical arguments, e.g., `subj:obj:obj`.
- The number of lexical heads must be identical to the number of arguments in the frame.
- All (nominal) lexical heads must be defined in WordNet (if they are "relevant" arguments, see below).

If a tuple is not defined according to the above restrictions, it is filtered from the input data.

- **`#clusters`**: number of clusters
- **`#iterations`**: number of training iterations
- **`init_probs`**: type of probability estimation

  The following four parameter values are allowed as types of probability estimation:

  - `random`: All probabilities in the `EMgraph` are initialised randomly.
  - `random_only_verbs`: Only the probabilities of the verbs in the clusters $p(v|c)$ are initialised randomly; all other probabilities are initialised uniformly.
  - `random_but_cluster`: All probabilities but the cluster probabilities $p(c)$ (which are initialised uniformly), are initialised randomly.
  - `random_items`: The probabilities of the verbs and the frames in the clusters, $p(v|c)$ and $p(f|c)$, are initialised randomly. All other probabilities are initialised uniformly.
- **`seed`**: seed for pseudo-random initialisation of probabilities, with $1 <= seed <= 1,000$
- **`const`**: constant for pseudo-random initialisation of probabilities, with $1 <= const <= RAND\_MAX$; for RAND_MAX, use `const=999`
- **`function_file`**: file with list of "relevant" argument types in frames, e.g., `subj, obj, pp-in`, etc.

  If the parameter value is `#`, there is no restriction on the types of arguments. Otherwise each line in the file contains an argument type that is taken into account.

  This definition of argument types allows the user to explicitly exclude arguments that are not relevant as nominal selectional preferences, e.g., adverbs, adjectives, particles, etc. Only arguments that are explicitly defined as "relevant" are used from the input file.
- **`freq_cut_off`**: minimum frequency for input tuples to be used in training
- **`wn_directory`**: directory that contains the WordNet files

  The WordNet files that are relevant for `LSCpref` are only `data.noun` and `index.noun`. However, the WordNet library functions also expect the following files (which might be empty to use `LSCpref`, though): `data.adj, data.adv, data.verb, index.adj, index.adv, index.verb, adv.exc, noun.exc, verb.exc, sents.vrb, sentidx.vrb`.

- **mdl_type**: type of MDL model, i.e., `cut` or `node`; NB: This manual only describes the `cut` mode, because the `node` mode has been re-implemented by PAC.

- **result_file**: name of file for output parameters of trained models

The parameter names are not relevant for `LSCpref` (i.e., column 1 in the parameter file is not read); what is relevant are the parameter values in column 2. An example parameter file (starting with line four) looks as follows.

```
input_file: test-tuples
#clusters: 20
#iterations: 50
init_probs: random_only_verbs
seed:         1
const:         5000000
function_file: args-test.txt
freq_cut_off: 10
wn_directory: WN-pref-files/EN-3.0
mdl_type: node
result_file: test-output
```

An example call using the above parameter file `test-params` could be
`LSCpref test-params >& test-output.log`.

During training, a log is written to standard output. In addition, the model parameters are written into
`test-output.params_#iteration`
after each 5th iteration, i.e., `test-output.params_5`, `test-output.params_10`, etc. The final parameter output is called `test-output.params`. The output parameters encode the transitions of the graph, using four columns per transition: start state of transition in column 1, end state of transition in column 2, transition probability in column 3, and transition frequency in column 4. This output can be transfered to an output format that is input to the GUI (described in Section 4.1.3) with the script `params2cluster` (described in Section 4.1.2).

4.1.2 params2cluster

`params2cluster` is a command-line tool that reads the parameter output from `LSCpref` (described in Section 4.1.1) and outputs probabilities of clusters, verbs and frames in clusters, and selectional preferences, in a format that can be used as input to the GUI (described in Section 4.1.3).

The tool requires two parameters:

- a parameter output file from `LSCpref`, cf. Section 4.1.1

- a method for how to calculate the selectional preferences:

  The selectional preferences for WordNet synsets in the trained models cannot directly be read from the model parameters. To calculate the selectional preferences for a specific cluster-frame-argument combination, for each WordNet synset that has been identified as relevant for this combination we compare the *path probability of the path between the WordNet synset and the WordNet top state in the selectional preference model* with the *path probability of the path between the same WordNet synset and the WordNet top state in the a priori model*. This comparison is done in three different ways:

  - `diff` calculates the difference between the two path probabilities,
  - `div` calculates the division between the two path probabilities, and
  - `skew` calculates the skew divergence between the two path probabiliites.

The output of the tool is automatically named `<params_file>_cluster_method`, with `<params_file>` the name of the parameter output file. For example, if the parameter output file was `test-output.params_5` and the method to calculate the selectional preferences was `diff`, the output of the parameter transfer is called `test-output.params_5_cluster_diff`.

An example call for `params2cluster` could be
`params2cluster test-output.params_5 diff`.

4.1.3 GUI: ClusterViewer

The GUI currently has to be started via command line specifying the file that will be displayed:

```
java ClusterViewer <name of cluster file>
```

4.2 Model 2 (EPS): *PAC*

The PAC implementation of the clustering models comprises the following set of programs:

- **pac-train** trains the model on verb-argument tuples and stores it in a binary format.
- **pac-print** prints the model in a human-readable form.
- **pac-prob** computes the probability of a tuple according to the clustering model
- **pac-clusterprob** computes the conditional probability of each cluster given the tuple and prints the most likely clusters and their probabilities.
- **pac-test** checks whether all probability distributions of a model sum up to 1 (with some tolerance for rounding errors).

- **pac-resume-training** can be used to continue a training run which was started with **pac-train**. This program has not been heavily used so far. The result of the resumed training cannot be expected to be exactly the same as the result of uninterrupted training because the probabilities of pruned transitions are replaced by back-off probabilities in the printed model.

The usage of the different programs and the file formats are documented in man pages of the programs. Run "man pac-train" for instance.

A typical call of the training program looks as follows:

```
> pac-train -vp -o 10 -c 50 -i 100 -s simple-slots wordnet tuples model
```

This command will train a PAC model with 50 clusters (-c 50) for 100 iterations (-i 100) using an MDL measure with variable precision (-vp) on the tuples stored in the file `tuples`. The resulting model is stored in the file `model`. Every ten iterations (-o 10), the current model is stored in a file called `model.tmpN` where `N` is the current iteration. The noun hierarchy for generalising selectional preferences is read from the file `wordnet`. The file `simple-slots` contains a list of slots whose heads are not nouns (such as `SBAR` or `ADJ`) and therefore cannot be generalised using WordNet.

The noun hierarchy is stored as follows:

```
0       GNROOT
        6998    7091    25455   32308   41234
1       Artefakt        Werk
        2       3       10      16      41      61      100     110     ...
2       Teil

3       Werkstck
        4       5       6       7
...
```

Each entry consists of two lines. The first line contains the synset index and the list of nouns contained in the synset, all of them separated by tab characters. The second line starts with a tab character and contains the list of hyponym indices.

The `tuples` file contains the tuples:

```
60      zunehmen        SUBJ    Wettbewerb
2       brauchen        SUBJ:OBJ        Staat   Waffe
12      helfen  SUBJ:VP-OC      Trick   <NONE>
...
```

Each line contains a frequency, a predicate (verb), a subcat frame, and a list or argument heads. All elements are separated by tab characters. The different frame slots are separated by colons (:). If some slot (such as `VP-OC`) is not associated with a head, the pseudo head `<NONE>` has to used instead.

The model is stored in a binary format. The `pac-print` program can be used to print it in human-readable form.

The size of the model grows with the number of different frames and the number of predicates (verbs). If the model training is too slow, it can help to reduce the number of different subcat frames and predicates by filtering the tuples.

Bibliography

Steven Abney and Marc Light. Hiding a Semantic Class Hierarchy in a Markow Model. In *Proceedings of the ACL Workshop on Unsupervised Learning in Natural Language Processing*, pages 1–8, College Park, MD, 1999.

Eneko Agirre and Philip Edmonds, editors. *Word Sense Disambiguation: Algorithms and Applications*. Springer-Verlag, 2006. URL: http://www.wsdbook.org/.

Marco Baroni and Adam Kilgarriff. Large Linguistically-processed Web Corpora for Multiple Languages. In *Proceedings of the 11th Conference of the European Chapter of the Association for Computational Linguistics*, Trento, Italy, 2006.

Leonard E. Baum. An Inequality and Associated Maximization Technique in Statistical Estimation for Probabilistic Functions of Markov Processes. *Inequalities*, III:1–8, 1972.

BNC. British National Corpus, 1995. URL `http://www.hcu.ox.ac.uk/BNC/`.

Carsten Brockmann and Mirella Lapata. Evaluating and Combining Approaches to Selectional Preference Acquisition. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 27–34, Budapest, Hungary, 2003.

Glenn Carroll and Mats Rooth. Valence induction with a head-lexicalized PCFG. In *Proceedings of the Third Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Granada, Spain, 1998.

Nathanael Chambers and Daniel Jurafsky. Improving the use of pseudo-words for evaluating selectional preferences. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 445–453, Uppsala, Sweden, July 2010. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P10-1046`.

Jinying Chen and Martha Palmer. Improving english verb sense disambiguation perform ance with linguistically motivated features and clear sense distincti on boundaries. *Language Resources and Evaluation*, 34 (43/2):181–208, 2009.

Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

Massimiliano Ciaramita and Mark Johnson. Explaining away Ambiguity: Learning Verb Selectional Preference with Bayesian Networks. In *Proceedings of the 18th International Conference on Computational Linguistics*, pages 187–193, Saarbrücken, Germany, 2000.

Stephen Clark and David Weir. Class-Based Probability Estimation using a Semantic Hierarchy. *Computational Linguistics*, 28(2):187–206, 2002.

I. Dagan, L. Lee, and F. Pereira. Similarity-based models of cooccurrence probabilities. *Machine Learning*, 34(1-3):43–69, 1999.

H. Dang and M. Palmer. Combining contextual features for word sense disambiguation. In *Proceedings of the SIGLEX/SENSEVAL Workshop on WSD: Recent Success and Future Direction*, Philadelphia, USA, 2002.

Bonnie J. Dorr and Doug Jones. Role of Word Sense Disambiguation in Lexical Acquisition: Predicting Semantics from Syntactic Cues. In *Proceedings of the 16th International Conference on Computational Linguistics*, pages 322–327, Copenhagen, Denmark, 1996.

Katrin Erk. A Simple, Similarity-based Model for Selectional Preferences. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic, 2007.

Christiane Fellbaum, editor. *WordNet – An Electronic Lexical Database*. Language, Speech, and Communication. MIT Press, Cambridge, MA, 1998.

John R. Firth. *Papers in Linguistics 1934-51*. Longmans, London, UK, 1957.

W. Gale, K.W. Church, and D. Yarowsky. Work on statistical methods for word sense disambiguation. In *AAAI Fall Symposium Series: Probabilistic Approaches to Natural Language (Working Notes)*, pages 54–60, 1992.

G. Harras, E. Winkler, S. Erb, and K.Proost. *Handbuch deutscher Kommunikationsverben, Teil 1: W"orterbuch*. Walter de Gruyter, Berlin, New York, 2004.

Zellig Harris. Distributional Structure. In Jerold J. Katz, editor, *The Philosophy of Linguistics*, Oxford Readings in Philosophy, pages 26–47. Oxford University Press, 1968.

Vasileios Hatzivassiloglou and Kathleen R. McKeown. Towards the Automatic Identification of Adjectival Scales: Clustering Adjectives According to Meaning. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 172–182, Columbus, OH, 1993.

Nancy Ide and Yorick Wilks. Making Sense about Sense. In Agirre and Edmonds (2006), pages 47–74. URL: http://www.wsdbook.org/.

Eric Joanis, Suzanne Stevenson, and David James. A General Feature Space for Automatic Verb Classification. *Natural Language Engineering*, 2008? To appear.

Adam Kilgarriff. Framework and results for english SENSEVAL. *Computers and Humanities*, 34 (1–2): 15–48, 2000.

Judith L. Klavans and Min-Yen Kan. The Role of Verbs in Document Analysis. In *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics*, pages 680–686, Montreal, Canada, 1998.

Philipp Koehn and Hieu Hoang. Factored Translation Models. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 868–876, Prague, Czech Republic, 2007.

Upali S. Kohomban and Wee Sun Lee. Learning Semantic Classes for Word Sense Disambiguation. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 34–41, Ann Arbor, MI, 2005.

Anna Korhonen. *Subcategorization Acquisition*. PhD thesis, University of Cambridge, Computer Laboratory, 2002. Technical Report UCAM-CL-TR-530.

Anna Korhonen, Yuval Krymolowski, and Zvika Marx. Clustering Polysemic Subcategorization Frame Distributions Semantically. In *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, pages 64–71, Sapporo, Japan, 2003.

Karim Lari and Steve J. Young. The Estimation of Stochastic Context-Free Grammars using the Inside-Outside Algorithm. *Computer Speech and Language*, 4:35–56, 1990.

Lillian Lee. On the Effectiveness of the Skew Divergence for Statistical Language Analysis. *Artificial Intelligence and Statistics*, pages 65–72, 2001.

Hang Li and Naoki Abe. Generalizing Case Frames Using a Thesaurus and the MDL Principle. *Computational Linguistics*, 24(2):217–244, 1998.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, 1999.

Paola Merlo and Suzanne Stevenson. Automatic Verb Classification Based on Statistical Distributions of Argument Structure. *Computational Linguistics*, 27(3):373–408, 2001.

Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional Clustering of English Words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190, Columbus, OH, 1993.

Detlef Prescher, Stefan Riezler, and Mats Rooth. Using a Probabilistic Class-Based Lexicon for Lexical Ambiguity Resolution. In *Proceedings of the 18th International Conference on Computational Linguistics*, Saarbrücken, Germany, 2000.

Philip Resnik. Selectional Preference and Sense Disambiguation. In *Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How?*, Washington, DC, 1997.

Jorma Rissanen. Modeling by Shortest Data Description. *Automatica*, 14:465–471, 1978.

Mats Rooth, Stefan Riezler, Detlef Prescher, Glenn Carroll, and Franz Beil. Inducing a Semantically Annotated Lexicon via EM-Based Clustering. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, Maryland, MD, 1999.

Michael Schiehlen. A Cascaded Finite-State Parser for German. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics*, pages 163–166, Budapest, Hungary, 2003.

Helmut Schmid. Probabilistic Part-of-Speech Tagging using Decision Trees. In *Proceedings of the 1st International Conference on New Methods in Language Processing*, 1994.

Helmut Schmid. Trace prediction and recovery with unlexicalized PCFGs and slash features. In *Proceedings of COLING-ACL'06*, pages 177–184, Sydney, Australia, 2006.

Sabine Schulte im Walde. Experiments on the Automatic Induction of German Semantic Verb Classes. *Computational Linguistics*, 32(2):159–194, 2006.

Sabine Schulte im Walde. Comparing Computational Approaches to Selectional Preferences: Second-Order Co-Occurrence vs. Latent Semantic Clusters. In *Proceedings of the 7th International Conference on Language Resources and Evaluation*, pages 1381–1388, Valletta, Malta, 2010.

Sabine Schulte im Walde, Christian Hying, Christian Scheible, and Helmut Schmid. Combining EM Training and the MDL Principle for an Automatic Verb Classification incorporating Selectional Preferences. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics*, Columbus, OH, 2008. To appear.

H. Schütze. Context space. In *AAAI Fall Symposium Series: Probabilistic Approaches to Natural Language (Working Notes)*, pages 113–120. Cambridge, MA, USA, 1992.

H. Seo, S. Lee, H. Rim, and H. Lee. Kunlp system using classification information model at senseval-2. In *proceedings of the Second International Workshop on Evaluating Word Sense Disambiguation Systems (SENSEVAL-2)*, Toulouse, France, 2001.

Eric V. Siegel and Kathleen R. McKeown. Learning Methods to Combine Linguistic Indicators: Improving Aspectual Classification and Revealing Linguistic Insights. *Computational Linguistics*, 26(4): 595–628, 2000.

Naftali Tishby, Fernando Pereira, and William Bialek. The Information Bottleneck Method. In *Proceedings of the 37th Annual Conference on Communication, Control, and Computing*, Monticello, IL, 1999.