Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE

Srđan Prodanović

Word Sense Disambiguation in Prague Dependency Treebank via Distributional Semantic Approach

Ústav formální a aplikované lingvistiky

Vedoucí diplomové práce:	RNDr. Jiří Hana, Ph.D., Institute of Formal and Ap
Studijní program:	Language and Communication Technologies
Studijní obor:	obor

Praha 2012

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V .... dne .... Podpis autora

Název práce: Word Sense Disambiguation in Prague Dependency Treebank via Distributional Semantic Approach

Autor: Srđan Prodanović

Katedra: Matematicko-fyzikální fakulta, Univerzita Karlova v Praze

Vedoucí diplomové práce: RNDr. Jiří Hana, Ph.D., Ústav formální a aplikované lingvistiky

Jméno a příjmení s tituly, pracoviště

Abstrakt: Distributional semantics is based completely on the distributional properties of words. The core resource is a single lexico–semantic lexicon that can be used for a variety of tasks, where word meanings are represented as vectors in Vector Space, and word similarities as distances between vector representations. Using strengths of similarities, appropriateness of terms given a particular context can be calculated and used for a variety of tasks, one of them being Word Sense Disambiguation. In this thesis several different approaches to models of Vector Space were examined and implemented in order to cross evaluate their performance on the Word Sense Disambiguation task in Prague Dependency Treebank.

Klíčová slova: word sense disambiguation, vector space model, prague dependancy treebank

Title: Word Sense Disambiguation in Prague Dependency Treebank via Distributional Semantic Approach

Author: Srđan Prodanović

Department: Matematicko-fyzikální fakulta, Univerzita Karlova v Praze

Supervisor: RNDr. Jiří Hana, Ph.D., Institute of Formal and Applied Linguistics

Abstract:

Keywords:

Contents

1. Introduction

1.1 About the task

Lexical disambiguation is concerned with determining the meaning of every word given their context. As a computational problem it is often described as "AI–complete", which means that in order to solve the problem of word meaning one needs to solve completely natural–language understanding or common– sense reasoning (Ide and Véronis 1998) . Fortunately, this is in not the case in the field of computational linguistics. There, this problem is generally called word sense disambiguation (WSD), and is defined as the problem of computationally determining which "sense" of a word is activated within a particular context. WSD is essentially a task of classification: word senses are the classes, a word's context contains necessary disambiguation information, and each occurrence of a word is assigned to one or more of its possible classes depending on the context information.

The Prague Dependency Treebank is a large annotated corpus built to further the research in Computational Linguistics for Czech language. It is fully annotated at morphological and syntactic analytic layers of annotation as well as in tectogrammatical layer. Like any other language, Czech language has its own share of words with multiple meanings. Thus the main task of this thesis is word sense disambiguation on PDT dataset.

1.2 About the model

Semantic theory has existed long enough to witness many different attempts to represent the meaning of words. Starting from Frege, Tarsky and Davidson's Propositional Logic representations where they concentrated on the Propositions about Symbols, calculating the "truth" of those Propositions. However, they were more focused on "truth conditions" and less on the content (representing what the propositions are about). (De Saussure 1878) in his first work talks about the *signifier* (the signs) and the *signified* (the "meaning"), followed with the theory of mediated reference (Frege 1892) where he makes a distinction between sense (intension) and reference (extension). Then comes the theory of direct reference (Rusell 1905) who equates the meaning with reference. If we look even further in history and beyond linguistic theory we can find examples in philosophy like with Aristotle where he describes concepts as a finite sets of features that belong to different categories (animate–inanimate, etc.). Leibnitz also used this approach when he tried to make his "language of meaning".

Some applications. A large part of Natural Language Processing today is concerned with tasks and applications related to the use of meaning, like lexicon acquisition, word–sense disambiguation, information access, machine translation, dialogue systems, etc. Vector space models have proven to be applicable in these fields. Another immediate application could be in the field of Information Retrieval, in particular for the expansion of user queries, since word–space models easily retrieve nearest neighbours in semantic space, in order to obtain better search results.

Intuitiveness. There is a certain intuitiveness behind statistical models of semantics. Experiments by Tom Mitchell[37] successfully predict the fMRI (fuctional MRI) brain images for particular nouns, based on fMRI images of seen nouns, completely relying on statistical approach. Lexical priming studies beginning with Ratcliff & McKoon (1978) and Swinney (1979) as well as eye movement studies (Rayner, Pacht, & Duffy, 1994) , suggest that ambiguous words first activate multiple interpretations, but very soon settle to that sense most appropriate to their discourse contexts. An analogy to estimating the amount of appropriateness of a word's meaning in some context could be drawn to the same way the scores of a word's meaning in some context are calculated, as will be presented in this thesis. It should be noted that computational models presented here are not necessarily similar to the way humans process information when they think about word meaning. They are however empirically consistent with human behaviour when they are performing semantic related tasks.

1.3 Thesis goals

Main goal of this Master's thesis is to, by utilizing various approaches to statistical representation of semantics, perform a Word Sense Disambiguation using Prague Dependency Treebank as a dataset. Various approaches that were used for the task utilize different statistical models of semantics, and they as well as subsume a variety of steps in linguistic and statistical preprocessing of text. All necessary parameters are tuned in order to achieve best results for every model examined. Therefore, the secondary goal of this thesis was to establish the methodology by which preprocessing steps and tuning of models parameters should occur.

1.4 Road map

This thesis is organized as follows: second chapter is describing the Word sense disambiguation task, its purpose, applications and overview of approaches. Next chapter explains the statistical model of semantics starting from motivation, rationale and then details the mathematical foundation of word–space models. Chapter after that is devoted to the problem of high dimensionality, present in word–space models, and gives an overview of best known techniques in overcoming this issue. Chapter four describes all the steps that can be taken when constructing the model, concentrating on presenting current approaches to such steps. With chapter four the theoretical part of the thesis is concluded. Chapter five, which is on experiments, details all the steps taken in experiments performed. Here are described: the research methodology and sections devoted to presenting tuning and results for three word–space models used for WSD task. Final chapters describe the implementation and in the user manual explain how the implementation should be run when performing experiments.

2. Computational model of semantics

The computational model of semantics presented here is referred to in the literature as the *word–space model* or *vector–space model* (VSM). Term was coined by Hinrich Schütze (1993) who defined the model as:

> *Vector similarity is the only information present in Word Space: semantically related words are close, unrelated words are distant. (p.896)*

2.1 Motivation

Despite the wealth of theories on meaning only a number of them has proven to be fully functional in an actual implementation. This is mainly due to the fact that linguistic data tends to be variable, inconsistent and vague. Keeping this in mind, a categorization of approaches can be viewed (Jurafsky, Martin 2000):

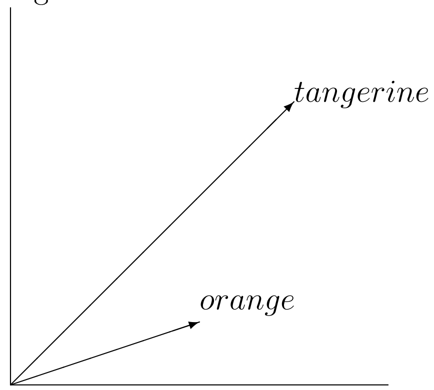
1. Representational approach: involves the creation of formal **meaning representations** that are intended to bridge the gap from language to commonsense knowledge of the world. These representations need to be able to support the practical aspects of semantic processing, like to resolve the truth of propositions, to support unambiguous representations, to represent variables, and to support inference. Human languages have various tools used to convey meaning, the most important being the one used to convey a **predicate–argument structure**. First order Predicate Calculus is a computationally tractable meaning representation language that is used mainly for the purpose of handling predicate–argument structure.

2. Syntax-driven approach: rests on the **Principle of Compositionality** that states that the meaning of a sentence can be composed from the meanings of its parts. Semantic analyzers that make use of static knowledge from the lexicon and grammar can create context-independent literal, or conventional, meanings. In Syntax-driven semantic analysis, the parts are the syntactic constituents of an input.

Vector–space models described in this thesis rely completely on the language data and do not have a single rule or constraint pre–written into the model. Apart from the advantage of avoiding laborious task of handcrafting rules for the model of semantics, statistical modeling has yet another advantage- its construction is entirely automatic, straight from the raw or annotated corpus. Since the modeling depends on the data set used in training, results are completely optimized toward that data set.

2.2 Similarity is proximity

The word–space model is a spatial representation of a word meaning. If every word is represented as a vector in n–dimensional space then the claim is that semantic (un)similarity of those words can be measured as a distance between their vectors. To illustrate this on a simple example for vectors representing meanings of 2 words in 2–dimensional space:



Semantic similarity between words can be measured as a distance between vectors representing *tangerine* and *orange*. Mathematical background of this claim will be explained in detail later in chapter 5.5.

Distance in space as a way to represent semantic similarity seems to be a very natural and intuitive idea. This has been pointed out in a number of works by George Lakoff and Mark Johnson (Lakoff & Johnson, 1980, 1999) where it is explained how humans use their spatio–temporal knowledge to conceptualize abstract objects. (Sahlgren 2006) notes that *similarity–is–proximity* also entails another geometric metaphor: *concepts–are–locations*. This is also important to observe because in Distributional Semantics word meanings are percieved according to the differential property of their geometric locations in the Word Space.

2.3 Distributional manifestation of meaning

In the word–space model similarities between words are automatically extracted from language data. As data, the word–space model uses statistics about the distributional properties of words. (Sahlgren 2006) has formulated from this insight his *Distributional Hypothesis* which states:

> *Words with similar distributional properties have similar meanings.*

There are number of examples in previous work to back this claim. (Schutze & Pedersen 1995) [21], claim that ”words with similar meanings will occur with similar neighbors if enough text material is available", and (Rubenstein & Goodenough 1965)[19] in one of the very first studies to explicitly formulate and investigate the distributional hypothesis state that "words which are similar in meaning occur in similar contexts" .

According to Zelig Harris in his "Mathematical Structures of Language" linguistic meaning is inherently differential, instead of referential which means that differences in meaning are visible through the differences in distribution. One of the first experiments to back this is (Rubenstein & Goodenough 1965)[19], who compared contextual similarities with synonymy judgments provided by university students. Their experiments demonstrated that there indeed is a correlation between semantic similarity and the degree of contextual similarity between words.

2.4 Context vectors

After describing the geometrical intuition behind the semantic similarity in this section will be explained how the model is built. Let us start by quoting Zelig Harris:

> The distribution of an element will be understood as the sum of all its environments. (Z. Harris, 1970, p.775)[58]

Let us illustrate on a one sentence example how a distributional model could be built from it:

> "Where there is smoke there is fire."

We start by determining what is the appropriate environment of the word. In linguistics word environment is called **context**. In our example we will restrict the context to the preceding and succeeding word. Distributions gathered from our example are shown in the table below.

	where	there	is	smoke	fire
where	0	1	0	0	0
there	1	0	2	0	0
is	0	2	0	1	1
smoke	0	0	1	0	0
fire	0	0	1	0	0

As we can see from the table every word is represented with a vector, for instance "smoke" is (0,0,1,0,0). To put it more formally: every vector is defined by n components, where each component represents a location in n–dimensional space. To quote Magnus Sahlgren's definition on context vectors:

> I call the co–occurrence counts *context vectors* because they effectively constitute a representation of the sum of the word's contexts.

2.4.1 A brief history of context vectors

The notion of context vectors has begun with the prominent work in psychology by Charles Osgood in the 1950's[6] on feature space representations of meaning, which he called *semantic differential approach to meaning representation.* In this approach words are represented as feature vectors where elements are contrastive adjective pairs such as "soft–hard", fast–slow", etc. The idea was to measure the psychological difference between words. An example from the research is given below.

Osgood's work was an inspiration for Stephen Gallant, who introduced the term "context vector" to describe the feature–space representations (Gallant, 1991a)[46]. In Gallant's algorithm, context vectors were defined with a set of manually derived features, such as "human, "man, "machine," etc. However as

	small–large	bald–furry	docile–dangerous
mouse	1	6	1
rat	2	6	4

Table 2.1: Feature vectors based on three contrastive pairs for words mouse and rat

some researchers later noticed drawbacks of this approach to describe the word's semantics were:
1. which features should be used?
2. how many features are enough?

These questions led to first approaches to *automatically* construct a feature space. One of the first approaches was by Gallant (1991b)[45] where he described the algorithm in two steps:

1. A context vector is initialized for each word as a normalized random vector.
2. While making several passes through the corpus, the context vectors are changed to be more like the context vectors of the surrounding words.

The results were then used for word–sense disambiguation, where he calculated meanings for words from context, and compared them to the manually defined ones (Gallant, 1991b)[45]. However, probably the most influential work comes from Hinrich Schütze (1992, 1993)[22][23], who built context vectors (which he calls "term vectors" or "word vectors") similarly to the approach described above: co–occurrence counts are collected in a words–by–words matrix, in which the elements record the number of times two words co–occur within a set window of word tokens. Context vectors are then defined as the rows or the columns of the matrix.

2.5 The co–occurrence matrix

The most formal definition of a co–occurrence matrix would be that it is a matrix of co–occurrence counts of its elements. The matrix can be a words–by–words matrix $w \times w$, where w are the word types in the data, or a words–by–documents matrix $w \times d$, where d are the documents in the data. A cell f_{ij} of the co–occurrence matrix contain the frequency of occurrence of word i in the context of word j or in document j, depending on the matrix type.
In the study on Vector Space Models of Semantics (Turney&Pantel, 2010)[53] authors look into various approaches to the task of semantic processing of text. They observe and classify Vector Space Models into three main classes, depending on the type of the co–occurrence matrix they are based on: term–document, word–context or pair–pattern matrices.

1. **Term–Document matrix:** The row vectors of the matrix correspond to terms (usually terms are words, but there are also other possibilities),

and the column vectors correspond to documents (web pages, for example). Term–Document matrix follows the "bag–of–words" hypothesis, which means that the order of words does not matter in order to estimate the importance of the query to the document (Salton et al. 1975) [15]. The "bag" here is like a mathematical set, with allowed duplicates. To illustrate this on a simple example using pseudo–words: bag {aa, ab, aa, ab, ac} contains elements aa, ab, ac. The order of bag's elements is of no relevance here– bag {aa, ab, aa, ab, ac} is the same as a bag {aa, aa, ab, ab, ac}. If we measure raw frequencies, this bag can be represented with a vector **A**=<2,2,1> where first element is the frequency of *aa*, second element is the frequency of *ab*, and third element is the frequency of *ac*. A set of bags can be represented as a matrix X, in which each column $x_{:j}$ corresponds to a bag, each row $x_{i:}$ corresponds to a unique term, and an element x_{ij} is the frequency of the i–th term in the j–th bag. If we look at the documents as bags, this example is easily transported into the Information Retrieval domain. The pattern of numbers in $x_{i:}$ is a kind of signature of the i–th term w_i; likewise, the pattern of numbers in $x_{:j}$ is a signature of the j–th document d_j. This means that the pattern of numbers reveal, to some degree, what the term or document is about.

The vector does not attempt to capture the structure in the phrases, sentences, paragraphs, and chapters of the document. Despite this omission, search engines work really well when based on term document matrix. Term–document matrix reflects in fact the similarity of documents. In the IR domain, search engines treat queries as documents, and return search results whose score is based on the degree of similarity between query vectors and all document vectors in corpus.

2. **Word–Context matrix:** Deerwester et al. (1990)[12] observed that we can shift the focus to measuring word similarity, instead of document similarity, by looking at row vectors in the term–document matrix, instead of column vectors. In general, we may have a word–context matrix, in which the context is given by words, phrases, sentences, paragraphs, chapters or documents. A word may be represented by a vector in which the elements are derived from the occurrences of the word in various contexts, such as windows of words (Lund & Burgess, 1996)[34], grammatical dependencies (Lin, 1998)[10], and richer contexts consisting of dependency links and selectional preferences on the argument positions (Erk & Padó, 2008)[26]. To illustrate this on a simple example: consider a co–occurrence matrix populated by simple frequency counting: if word i co–occurs 16 times with word j, we enter 16 in the cell f_{ij} in the words–by–words co–occurrence matrix. The co–occurrences are normally counted within a context window spanning some (usually small) number of words.

3. **Pair– Pattern Matrix:** reflects the similarity of relations. Row vectors correspond to pairs of words, such as *mason : stone* and *carpenter : wood*, and column vectors correspond to the patterns in which the pairs co–occur, such as *"X cuts Y"* and *"X works with Y"*. Turney et al. (2003) introduced the use of the pair–pattern matrix for measuring the semantic similarity of relations between word pairs, which in this case is the similarity of row

vectors. The *latent relation hypothesis* is that pairs of words that co–occur in similar patterns tend to have similar semantic relations (Turney, 2008a). Word pairs with similar row vectors in a pair–pattern matrix tend to have similar semantic relations.

(Schütze and Pedersen 1993)[20] defined two ways that words can be distributed in a corpus of text: If two words tend to be neighbours with each other, then they are syntagmatic associates. If two words have similar neighbours, then they are paradigmatic parallels. As they noted, syntagmatic associates are often different parts of speech, whereas paradigmatic parallels are usually the same part of speech. Syntagmatic associates tend to be semantically associated (for example, bee and honey), while paradigmatic parallels tend to be taxonomically similar (for example doctor and nurse).

While word–document and word–word matrices reflect attributional relations between word senses, the pair–pattern matrix reflects relational similarity between word senses. This distinction was explained in (Gentner 1983)[16]. Being that we are focused in this dissertation on the application of vector space models solely on the task of WSD we are interested only in models that stem from attributional kind of relationship, thus utilizing the first two kinds of matrices. The pair–pattern matrix has proven track record with other applications, for example one of them being the analogies task [52], and is therefore of no interest to implement in this research.

3. Word sense disambiguation

3.1 Word sense(s)

A meaning of a word is a variable category, very much dependent on its surrounding context. In the field of lexical semantics it is found that many words have overlapping or extended meanings (Kilgarriff 1997)[2]. ”Polysemy" of a word means that it has multiple, but related meanings. At a more coarse grained level of meaning words that have the same spelling and pronunciation as each other but different meanings and origins are called *homonyms*. Polysemy is a feature of words while ”ambiguity" is a property of text. If there is uncertainty about the meaning of a word there is ambiguity about the whole text as well. Probably the most well known example of an ambiguous word is the word *bank*, which contains two homonyms: as a as financial institution, and as a river bank. Bank as financial institution splits further into the following cloud of related senses: the company or institution, the building itself, the counter where money is exchanged, a fund or reserve of money, a money box (piggy bank), the funds in a gambling house (WordNet 2.1)[1]. Different words have different number of meanings. A view on the number of senses word might have is somewhat controversial. Some argue that task– independent senses simply cannot be enumerated in a list (Kilgarriff 1997)[2] while others claim that words can have only a single, abstract meaning (Ruhl 1989)[8].

WSD was first formulated as a distinct computational task during the early days of machine translation. It was Weaver (1949)[55] who noted in his memo on machine translation the importance of context in determining word sense(s), by conducting a simple experiment- observing words from a sentence in isolation, and trying to determine their meaning. (Zipf 1949)[17] published his "Law of Meaning" where he states that more frequent words have more senses than less frequent words in a power–law relationship. A valuable point worth noting in any experiment with WSD. This was later confirmed for the British National Corpus (Edmonds 2005) [42] as well.

Statistical and machine learning methods have been successfully applied to the WSD problem. Methods that train on manually sense–tagged corpora (like Penn Treebank, or Prague Dependency Treebank) are called supervised learning methods. Since models implemented in this thesis are trained on annotated corpus, they are as well supervised. Supervised methods have become in general the mainstream approach to WSD, with the best results in all tasks of the Senseval [2] competitions.

[1] http://wordnet.princeton.edu/
[2] http://www.senseval.org/

3.2 Applications of WSD– why is it important?

WSD's importance lies in the fact that it enables other tasks and applications of computational linguistics (CL) and natural language processing (NLP) such as information retrieval(IR), parsing, machine translation(MT), semantic interpretation, text mining, and (lexical) knowledge acquisition. Detail applications of WSD to each of these fields is presented below.

Machine translation (MT). In MT, choice of the correct translation of a word is probably one of the hardest tasks in that field. WSD originally introduced for lexical choice of words that have multiple translations depending on their context. For example, in an English–French financial news translator, the English noun *change* could translate to either *changement* ('transformation') or *monnaie* ('pocket money'). Nowadays, there is contrasting evidence that WSD can benefit MT: for instance, (Carpuat and Wu 2005) claimed that WSD cannot be integrated into MT applications, while (Dagan and Itai 1994) show that the proper use of WSD leads to an increase in the translation performance.

Information retrieval (IR). In IR, documents retrieved depend on query terms, which are often ambiguous. For instance, given the query "depression" should the system return documents about illness, weather systems, or economics? Current IR systems do not use explicit WSD, and rely on the user typing enough context in the query to only retrieve documents relevant to the intended sense (i.e., "tropical depression"). (Sanderson 1994)concluded that with queries containing large number of words, WSD cannot benefit IR. (Schutze and Pedersen 1995) have however demonstrated that at a 90% accuracy level of WSD, there is an IR performance improvement by about 4,3%(from 29.9% to 34.2%).

Information extraction (IE) and text mining. WSD is required for the accurate analysis of text in many applications. For example if we wish to somehow mark the difference between *medical drugs* and *illegal drugs* we would need to have these phrases disambiguated first. More generally, the Semantic Web requires automatic annotation of documents according to a reference ontology: all textual references must be resolved to the right concepts and event structures in the ontology (Dill et al. 2003)[50] (Narayan et al. 2010) [38].

Lexicography. Modern lexicography is corpus–based, thus WSD and lexicography can work for each other. WSD is providing empirical sense grouping and contextual indicators of sense to lexicographers, who provide better sense inventories and sense–annotated corpora to enable better WSD. Some of the more famous examples are the HECTOR project (Atkins 1993) and the Sketch Engine (Kilgarriff et al. 2004).

However it should be noted that explicit WSD by itself, has not always demonstrated decisive benefits in real end–to–end applications (Navigli 2009:51). There have been isolated results that show some improvements, but just as often WSD can hurt performance, as is the case in one IR experiment (Sanderson 1994).

3.3 Classification of approaches to WSD

In a broad view, we can distinguish two main streams of approaches to WSD (Navigli 2009). First stream classifies WSD approaches based on level of supervision it requires:

1. supervised: use machine learning techniques to learn a classifier from labeled training sets of any kind

2. unsupervised: employ mainly clustering techniques on unlabeled corpora, and do not employ a manually sense-tagged corpus to provide a sense choice for a word in context.

Another way of viewing approaches to WSD is according to the nature of the resource used during training. Methods that rely primarily on dictionaries, thesauri, and knowledge bases (like ontologies), without using any corpus evidence, are called *dictionary–based* or *knowledge–based*. Contrary to them there are methods that work directly with raw, unannotated corpora, and are termed as *corpus–based methods*.
Senseval (later Semeval) is a competition among NLP practitioners devoted solely to WSD and WSD– related tasks. Two oldest variants of WSD task is based on the content of the test set used for evaluation of WSD systems entered in that competition:

1. Lexical sample task: a system is required to disambiguate a restricted set of target words usually occurring one per sentence.

2. All-words WSD task: systems are expected to disambiguate all open-class words in a text (i.e., nouns, verbs, adjectives, and adverbs)

Since the inception of Senseval systems perform better on the lexical sample task.

3.3.1 Unsupervised learning methods

Unsupervised methods have the potential to overcome the new knowledge acquisition bottleneck (manual sense–tagging), which is the lack of large-scale resources manually annotated with word senses. These methods are able to induce word senses from training text by clustering word occurrences, and then classifying new occurrences into the induced clusters/senses. They are based on the idea that the same sense of a word will have similar neighboring words. However, they may not discover clusters equivalent to the traditional senses in a sense inventory. While supervised WSD is typically identified as a sense labeling task, that is, the explicit assignment of a sense label to a target word, unsupervised WSD performs word sense discrimination, which aims to divide “the occurrences of a word into a number of classes by determining for any two occurrences whether they belong to the same sense or not [Schútze 1998, page 97]. Main approaches to unsupervised WSD are:

1. Context Clustering: Each occurrence of a target word in a corpus is represented as a context vector. The aim of the approach is to cluster context vectors, that is, vectors which represent the context of specific occurrences

of a target word. Sense discrimination can then be performed by grouping these context vectors using a clustering algorithm. (Schutze 1998) proposed an algorithm, called context-group discrimination, which groups the occurrences of an ambiguous word into clusters of senses, based on the Expectation Maximization algorithm. A different clustering approach consists of agglomerative clustering (Pedersen and Bruce 1997). Initially, each instance constitutes a singleton cluster. Next, agglomerative clustering merges the most similar pair of clusters, and continues with successively less similar pairs until a stopping threshold is reached.

2. Word Clustering: Aim at clustering words which are semantically similar and can thus convey a specific meaning. One approach to word clustering (Lin 1998a) involves identification of words similar (possibly synonymous) to a target word. To discriminate between the senses, a word clustering algorithm is applied to induce senses of the target word. In the next attempt, *clustering by committee* (CBC) algorithm (Lin and Pantel 2002), a different word clustering method was proposed. To calculate the similarity each word is represented as a feature vector, where each feature is the expression of a syntactic context in which the word occurs. Recursive procedure is applied to cluster word into *committees*, and then discrimination is performed based on the similarity of feature vector to the centroid of each committee.

3.3.2 Supervised methods

Supervised methods are also called *exemplar based* methods on account of the learning phase. The training set used to learn the classifier typically contains a set of examples in which a given target word is manually tagged with a sense from the sense inventory of a reference dictionary. They are predominantly consisting of Machine Learning(ML) methods.

1. Probabilistic Methods: these methods usually estimate a set of probabilistic parameters that express the conditional or joint probability distributions of categories and contexts (described by features). These parameters can be then used to assign to each new example the particular category that maximizes the conditional probability of a category given the observed context features.

2. Methods Based on the Similarity of the Examples: the methods in this family perform disambiguation by taking into account a similarity metric. This is done by comparing new examples to a set of learned vectors (one for each word sense) and assigning the sense of the most similar vector, or by searching in a stored base of annotated examples for the most similar examples and assigning the most frequent sense among them. VSM fall into this family of approaches.

3. Methods Based on Discriminating Rules: decision lists and decision trees use selective rules associated with each word sense. Given an example to classify, the system selects one or more rules that are satisfied by the example features and assign a sense based on their predictions.

4. Linear Classifiers and Kernel-Based Approaches: Linear classifiers have been very popular in the field of information retrieval (IR), since they have been used successfully as simple and efficient models for text categorization. A linear (binary) classifier is a hyperplane in an n-dimensional feature space that can be represented with a weight vector w and a bias b indicating the distance of the hyperplane to the origin. In the WSD task it classifies meanings of polysemous word against its context.
   Support Vector Machines (SVM) is the most popular kernel-method. The learning phase consists of choosing the hyperplane that separates the positive examples from the negatives with maximum margin between. This learning bias has proven to be very powerful and has lead to very good results in many pattern recognition, text, and NLP problems.

3.4 Approaches that use VSM for WSD

As mentioned in the previous section Vector Space Model (VSM) approaches fall under methods based on the similarity of the examples. There are many ways to calculate the similarity between two examples. Assuming the VSM one of the simplest similarity measures is to consider the angle that both example vectors form (i.e., the cosine measure). (Leacock et al. 1993)[32] compared VSM to ML techniques such as Neural Networks, and Naive Bayes methods, and drew the conclusion that the two first methods slightly surpass the last one in WSD.

There was also an automatic and unsupervised approach (Schütze 1998)[24], based on clustering. Senses are interpreted as groups (or clusters) of similar contexts of the ambiguous word. Words, contexts, and senses are represented in Word Space, a high-dimensional, real-valued space in which closeness corresponds to semantic similarity. The algorithm is unsupervised in both training and application: senses are discriminated by learning from a corpus without labeled training instances or other external knowledge sources.

Another unsupervised approach at WSD done by (Pantel & Lin, 2002a) [40] is a clustering algorithm called CBC (Clustering By Committee). The centroid of the members of a committee is used as the feature vector of the cluster. Words to are assigned to their most similar clusters. After assigning an element to a cluster, overlapping features of the cluster are removed from that element. Finally, each cluster that a word belongs to represents one of its senses. Weighting is performed with PMI(explained in the chapter 5.4.3) and distance between vectors is calculated using cosine distance.

3.5 State–of–the–art

In 1997, Senseval–1 (Kilgarriff and Palmer 2000)[3] found accuracy of 77% on the English lexical sample task, just below the 80% level of human performance (estimated by inter–tagger agreement). In 2001, scores at Senseval–2 (Edmonds and Cotton 2001)[43] appeared to be lower, but the task was more difficult, as it was based on the finer grained senses of WordNet. The best accuracy on the

English lexical sample task at Senseval–2 was 64% (to an inter–tagger agreement of 86%). Senseval–2 showed that supervised approaches had the best overall performance.
By 2004, the top systems on the English lexical sample task at Senseval–3 (Mihalcea and Edmonds 2004) [1] were performing at human levels according to inter–tagger agreement. The top ten systems, (all supervised) made between 71.8% and 72.9% correct disambiguations compared to an inter–tagger agreement of 67%. The best unsupervised system overcame the most–frequent–sense baseline achieving 66% accuracy.

4. Word space implementations

We have described in the previous chapters the intuitions behind Word Space Models in Distributional Semantics as well as how the semantic similarity between words represented there can be computed. This chapter will be dedicated to presenting how some of the best known Word Space Models are implemented with respect to difficulties that arise when dealing with corpora of large volume. The first section will outline some of the main difficulties that stem from facing large and sparse data. Sections that follow will present main implementations of Word Space Models that address these difficulties. These models will also be reviewed with respect to their applicability on the task of WSD.

4.1 High dimensionality and data sparseness

The word–space methodology relies on statistical evidence to construct the word space. If there is not enough data, there is no required statistical foundation to build a model of word distributions. At the same time with a substantial volume of data, context (co–occurrence) matrices easily become very large, with high dimensional context vectors which seriously affects the scalability and efficiency of the algorithm. As we can already see this issue presents a problem that is fortunately, neither new nor unsolvable. After all, a Word Space Model implemented by the vast majority of commercial Search Engines handles this issue gracefully by approaching it from its technical side– the most common solution is division and distribution of index. However, in this thesis we are interested in addressing this problem from another perspective, and see how this issue can be resolved from the mathematical stance to prevent the models to become computationally expensive.

Another issue that appears within a high–volumed corpus is that the majority of its term vectors turns out as sparse. This means that most the elements of a term vector are zero, while only a small amount of elements has non–zero values. This fact stems from the notion that only a tiny amount of the words in language are distributionally promiscuous; the vast majority of words only occur in a very limited number of contexts. This phenomenon is well known, and is an example of the general Zipf's law (Zipf, 1949)[17]. This reflects on the co-occurrence matrix, causing the majority of term vectors from it to be sparse.

4.2 Dimensionality reduction

The solution to both issues mentioned in the previous section is an NLP operation (borrowed from linear algebra) called *dimensionality reduction*. Dimensionality reduction represents high–dimensional data in a low–dimensional space, so that both the dimensionality and sparseness of the data are reduced, while still retaining as much of the original information as possible. There are two general ways to perform dimensionality reduction, that can also be combined: word filtering and matrix reduction.

The simplest way to perform dimensionality reduction is to filter out words and documents based on either linguistic or statistical criteria (Sahlgren 2006). Linguistic criteria can be word's affiliation to certain (unfavorable) grammatical classes. Statistical criteria can be an undesirable statistical property of a word. Both linguistic and statistic criteria are discussed in more detail in the preprocessing section. In the following sections three major, and most influential approaches in performing dimensionality reduction by reducing the co–occurrence matrix are presented.

4.3 Latent Semantic Analysis

Probably the best know VSM that performs dimensionality reduction is Latent Semantic Analysis (LSA) (Landauer & Dumais, 1997)[30]. LSA was developed under the name Latent Semantic Indexing (LSI) (Dumais et al., 1988; Deerwester et al., 1990)[14][12] in the late 1980s as an extension to the traditional vector–space IR. The terms LSI and LSA have since become more or less synonymous in the literature, though LSI is a term more frequently used in the IR domain. The development of LSA was motivated by the inability of the vector–space model to handle synonymy: a query about "boats" will not retrieve documents about "ships" in the standard vector–space model. LSA addresses this problem by reducing the original high–dimensional vector space into a much smaller space, in which the original dimensions that represented words and documents have been shrunk into a smaller set of latent dimensions that collapses words and documents with similar context vectors.

The dimensionality reduction is accomplished by using a statistical dimensionality–reduction technique called Singular Value Decomposition (SVD). (Golub and Kahan 1965)[18] introduced SVD as a decomposition technique for calculating the singular values, pseudo–inverse and rank of a matrix. The equation is given below:

$$A = USV^T \tag{4.1}$$

,where:
U is a matrix whose columns are the eigenvectors[1] of the AA^T matrix. These are called left eigenvectors.
S is a matrix whose diagonal elements are the singular values of A. This is a diagonal matrix[2], so its non–diagonal elements are zero
V is a matrix whose columns are the eigenvectors of the A^TA matrix. These are called the right eigenvectors.
V^T is the transpose matrix[3] of matrix V.

[1]The eigenvectors of a square matrix are the non–zero vectors that, after being multiplied by the matrix, remain parallel to the original vector.

[2]diagonal matrix is a matrix (usually a square matrix) in which the entries outside the main diagonal are all zero

[3]The transpose of a m by n matrix is defined to be a n by m matrix that results from interchanging the rows and columns of the matrix.

This outlines the basic mechanism of LSA when performing SVD. Main steps which are performed in building and using LSA are:

- Building a term–document matrix;
- SVD based dimensionality reduction
- Calculating the cosine measure to compute vector similarities between vectors from the co–occurrence matrix

LSA is well applied in information retrieval(Deerwester et al., 1990; Dumais, 1993; Dumais et al., 1997; Jiang & Littman, 2000). The incentive for using SVD in an information–retrieval setting is obvious: words with similar co–occurrence patterns are grouped together, allowing for the retrieval of documents that need not necessarily contain any of the query words.

In the domain of the Contextual Disambiguation LSA has had both good and bad performances, depending on the aspect of the task(Landauer & Dumais, 1997)[30]. Good performance is demonstrated for the task of choosing the correct one out of all meanings of a polysemous word, when presented as certain context. On the other hand, for polysemous words that take many semantically diverse meanings, it has proven to be ineffective when it comes to acquisition and representation of multiple separate meanings of a single word.

4.4 Hyperspace Analogue to Language

A pretty different word–space implementation is the Hyperspace Analogue to Language (HAL) (Lund et al., 1995)[33], which in contrast to LSA was developed specifically for word–space research. HAL builds a words–by–words co–occurrence matrix, which is populated by counting word co–occurrences within a directional context window 10 words wide. The co–occurrences are weighted with the distance between the words, so that words that occur next to each other get the highest weight, and words that occur on opposite sides of the context window get the lowest weight. The result of this operation is a directional co–occurrence matrix in which the rows and the columns represent co–occurrence counts in different directions.

Each row–column pair (i.e. the left and right–context co–occurrences) is then concatenated to produce a very–high–dimensional context vector, which has a dimensionality two times the size of the vocabulary. If such very–high–dimensional vectors prove to be too costly to handle, HAL reduces their dimensionality by computing the variances of the row and column vectors for each word, and discarding the elements with lowest variance, leaving only the 100 to 200 most variant vector elements. In HAL, two words are semantically related if they tend to appear with the same words. Summarized, HAL is built in the following steps:

- Building a directional words–by–words matrix.
- Distance weighting of the co–occurrences.
- Concatenation of row–column pairs.
- Normalization of the vectors to unit length.
- Minkowski metric to compute vector similarities.

The Minkowski metric, also called the Minkowski tensor or pseudo-Riemannian metric, is a tensor $\eta_{\alpha\beta}$ whose elements are defined by the matrix:

$$(\eta)_{\alpha\beta} = \begin{vmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

,which means that for 2 vectors α and β when computing similarity between them, the first coordinate is taken with a negative sign. Minkowski metric is normally used in physics, in theory of relativity, and there the first coordinate represents the time dimension.

4.5 Random Indexing

Another approach is Random Indexing (RI) (Kanerva et al., 2000; Karlgren & Sahlgren, 2001), developed at the Swedish Institute of Computer Science (SICS) based on Pentti Kanerva's work on sparse distributed memory (Kanerva, 1988)[27]. Like the previous approaches described in this chapter RI is motivated as well with the problem of high dimensionality. What is different with RI is how it addresses that problem: while previous approaches make lower–dimensional context vectors which are easier to compute with, they do not solve the problem of initially having to collect a potentially huge co–occurrence matrix. Even implementations that use powerful dimensionality reduction, such as SVD, need to initially collect the words–by–documents or words–by–words co–occurrence matrix. RI addresses this problem at its source, and removes the need for the huge co–occurrence matrix.

RI *incrementally accumulates context vectors* which can then, if needed, be assembled into a co- occurrence matrix (both words–by–documents and a words–by–words). RI accumulates context vectors in a two–step operation:

1. Each context (i.e. each document or each word type) in the text is assigned a unique and randomly generated representation called an *index vector*. In RI, these index vectors are sparse, high-dimensional, which means that their dimensionality r is on the order of thousands, and that they consist of a small number of randomly distributed non–zero elements (as many +1s as –1s). Each word also has an initially empty context vector of the same dimensionality r as the index vectors.

2. The context vectors are then accumulated by advancing through the text one word token at a time, and adding the context's (the surrounding word types' or the current document's) r–dimensional index vector(s) to the word's r–dimensional context vector. When the entire data has been processed, the r–dimensional context vectors are effectively the sum of the word's contexts.

If we then want to construct the equivalent of a co-occurrence matrix, we can simply collect the r-dimensional context vectors into a matrix of order $w \times r$, where w is the number of unique word types, and r is the chosen dimensionality for vectors. The dimensions in the RI vectors are randomly chosen, and therefore do not represent any kind of context (which is the case with the original co–occurrence matrix). Furthermore, r is chosen to be much smaller than the size of word types and the number of documents in the data, which means that RI will accumulate (roughly) the same information in the $w \times r$ matrix as other word-space implementations collect in the $w \times w$ or $w \times d$ co-occurrence matrices, except that in case of RI $r \ll d; w$.

If we think about every document in the term–document matrix to be different from each other, we can represent them with n–dimensional unary, zero vector, that have 1 written in a different place. These vectors are orthogonal[4].

[4]Two vectors, $\vec{x}$ and $\vec{y}$ are orthogonal if their dot product is zero. Dot product is an algebraic operation that takes two equal–length sequences of numbers (usually coordinate vectors) and

The r-dimensional random index vectors are only nearly orthogonal. This near–orthogonality of the random index vectors is in fact an explanation of the RI methodology. There are many more nearly orthogonal than truly orthogonal directions in a high–dimensional space (Kaski, 1999)[47], and choosing random directions, as is it is done with index vectors, can *approximate* orthogonality. The near-orthogonality of random directions in high-dimensional spaces is exploited by a number of dimensionality-reduction techniques which include methods such as Random Projection (Papadimitriou et al., 1998)[41], Random Mapping (Kaski, 1999)[47], and Random Indexing. The dimensionality of a given matrix F can be reduced to F' by multiplying it with (or projecting it through) a random matrix R:

$$F'_{w \times r} = F_{w \times d} R_{d \times r} \tag{4.2}$$

Obviously, the choice of the random matrix R is an important design decision. If the d random vectors in matrix R are orthogonal, so that $R_T R = I$, then $F' = F$. If random vectors are nearly orthogonal, then $F' \approx F$ in terms of the similarity of their rows. RI uses the following distribution for the elements of the random index vectors:

$$r_{ij} = \begin{cases} +1 & \text{with probability } \frac{\epsilon/2}{r} \\ 0 & \text{with probability } \frac{r-\epsilon}{r} \\ -1 & \text{with probability } \frac{\epsilon/2}{r} \end{cases} \tag{4.3}$$

where r is the dimensionality of the vectors, and ϵ is the number of non-zero elements in the random index vectors, usually a very small number.

Random Indexing is comparably robust with regards to the choice of parameters. Other word-space implementations, such as LSA, are very sensitive to the choice of dimensionality for the reduced space. For RI, the choice of dimensionality is a trade–off between efficiency and performance (random projection techniques perform better the closer the dimensionality of the vectors is to the number of contexts in the data (Kaski, 1999; Bingham & Mannila, 2001))[47][13]. Performance of RI reaches a stable level when the dimensionality of the vectors become sufficiently large as concluded in (Sahlgren & Karlgren, 2005a)[28].

returns a single number obtained by multiplying corresponding entries and then summing those products.

5. The theory behind experiments

In previous chapters we have explained the intuitions and approaches in building Semantic Vector Models. This chapter is devoted to outlining possible approaches in preprocessing, matrix normalization and evaluation with respect to experiments performed in this thesis on the of Word Sense Disambiguation (WSD) in Prague Dependency Treebank1 (PDT).

5.1 Linguistic preprocessing

Before generating a term–document or a word–context matrix it can be useful to apply some sort of linguistic preprocessing to the raw text. First type of linguistic preprocessing constitutes of *text normalization*, where words can be filtered out based on their Part–Of–Speech (POS)[1], then stemming or lemmatization and finally *annotation*.

POS filtering normally consists of removing words that belong to closed grammatical classes[2], since they are assumed to have little or no semantic meaning. However POS filtering removes only a small part of the lexicon, because the majority of words belong to open grammatical classes.

In linguistic morphology and information retrieval (IR), stemming is the process of reducing inflected (or sometimes derived) words to their stem, base or root form. Inflection characterizes a change in word form when found in a different case, gender, number, voice, tense, person or mood. In IR all variants of a word should be considered as just a single word. In English, affixes are simpler and more regular than in many other languages, and stemming algorithms based on heuristics (rules of thumb) work relatively well (Porter, 1980; Minnen, Carroll, & Pearce, 2001)[44]. The most popular stemming approach is the language independent Porter's stemmer. Lemmatization is the process of assigning a form its correct lemma (canonical/base/dictinary form). The difference between stemming and lematization can be illustrated on the following example: word "better" is an inflected form for comparison of adjective "good". This link is missed by stemming, as it requires a dictionary look–up. Lemmatization utilizes the use of a vocabulary and morphological analysis of words, and attempts to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma*. However, lemmatization is a difficult task– especially for highly inflected natural languages having a lot of words for the same normalized word form, which is in fact the case with the Czech language. Another difference between lemmatization and stemming is that stemming sometimes collapses derivationally related words, whereas lemmatization commonly only collapses the

[1]In grammar, a part of speech is a linguistic category of words (or more precisely lexical items), which is generally defined by the syntactic or morphological behaviour of the lexical item in question

[2]Closed grammatical classes rarely acquire new members, like adpositions, pronouns, conjunctions, and determiners. Open grammatical classes, on the other hand, and constantly drop, replace, and add new members. Examples of such classes are nouns, adjectives, and verbs.

different inflectional forms of a lemma.

Annotation is the process that is inverse of normalization. Just as different strings of characters may have the same meaning, it also happens that identical strings of characters may have different meanings, depending on the context. Common forms of annotation include part–of–speech tagging (marking words according to their parts of speech), word sense tagging (marking ambiguous words according to their intended meanings), and parsing (analyzing the grammatical structure of sentences and marking the words in the sentences according to their grammatical roles) (Manning & Schütze, 1999)[35]. In our experiments we are using a treebank as a dataset, which is as resource fully annotated on morphological, syntactics and semantic level the task of annotation comes down to simply extracting the desired tag connected to the word. Since we are only using semantic annotation I will just mention here few more examples that use semantic annotation: disambiguating abbreviations in queries (Wei,Peng, & Dumoulin, 2008)[56] and finding query keyword associations (Lavrenko & Croft, 2001)[31] and (Cao, Nie, & Bai, 2005)[4].

5.2 Statistic preprocessing

Statistic preprocessing is a process of filtering words that have an undesirable statistical property, like very high or very low frequency of occurrence. It should be noted that filtering high frequency words has approximately the same effect as POS filtering, due to the generally low count of such words(Zipf, 1949)[17]. More sophisticated statistical criteria for filtering includes filtering based on the TFIDF, and different variants and mixtures of the Poisson distribution (Katz, 1996) [29].

5.3 Types and Tokens

After preprocessing phase the time is to decide whether to base the co–occurrence matrix on types or tokens. A token is a single instance of a symbol, whereas a type is a general class of tokens (Manning et al., 2008)[36]. Consider the following example built on three short sentences taken from PDT:

(5.1) Ale šance je přesto minimální.

(5.2) Dnes je také mnohem menší šance.

(5.3) Přesto se stávat.

In this example there are eleven types and fourteen tokens present. Types "šance", "je" and "přesto" each have 2 tokens in this mini–corpus, that consists of 3 documents, if we consider every sentence here a document. We can represent this example with a token–document matrix or a type–document matrix. The token–document matrix has fourteen rows, one for each token, and three columns, one for each line (Figure 6.1). The type–document matrix has eleven rows, one for each type, and three columns (Figure 6.2). A row vector in the token matrix has binary values: an element is 1 if the given token appears in the given document and 0 otherwise. A row vector for a type has integer values: an element is the frequency of the given type in the given document. A type vector is the sum of the corresponding token vectors.

	Ale šance je přesto minimální	Dnes je také mnohem menší šance	Přesto se stávat
Ale	1	0	0
šance	1	0	0
je	1	0	0
přesto	1	0	0
minimální	1	0	0
Dnes	0	1	0
je	0	1	0
také	0	1	0
mnohem	0	1	0
menší	0	1	0
šance	0	1	0
Přesto	0	0	1
se	0	0	1
stávat	0	0	1

Figure 5.1: Example of token–document matrix

	Ale šance je přesto minimální	Dnes je také mnohem menší šance	Přesto se stávat
Ale	1	0	0
šance	1	1	0
je	1	1	0
přesto	1	0	1
minimální	1	0	0
Dnes	0	1	0
také	0	1	0
mnohem	0	1	0
menší	0	1	0
se	0	0	1
stávat	0	0	1

Figure 5.2: Example of type–document matrix

In applications dealing with polysemy, one line of approaches uses vectors that represent word tokens (Schütze, 1998)[24] while others use vectors that represent word types (Pantel & Lin, 2002)[40]. Typical word sense disambiguation (WSD) algorithms deal with word tokens (instances of words in specific contexts) rather than word types, although a defining characteristic of the VSM is that it is concerned with co-occurrence counts. In our experiments we will be using both types of vectors, although the majority of experiments will be based on the type vectors.

5.4 Building the matrix

When the basic unit of frequency counting is chosen (as explained in the previous section), and (optional) preprocessing is done on the corpus, the co–occurrence matrix can be built. As outlined in the chapter 2.5 the matrix can be either a word–document or word–context, based on their applicability on the task of WSD (therefore, the pair–pattern matrix was ruled out). When all frequencies are counted and placed in the matrix, frequency counts are normalized, and/or matrix's dimensionality can be reduced (using the models described in chapters 4.3 and 4.5).

5.4.1 Normalizing the frequency counts

When frequency counts are calculated for every element in the matrix, it customary to weight the elements in the matrix. The hypothesis is that surprising events, if shared by two vectors, are better signs of the similarity between these vectors than less surprising events. For example, when measuring the semantic similarity between the words *cat* and *puss*, the contexts like *fur* and *pat* are more discriminative of their similarity than the contexts *have* and *like*. The idea stems way back to Information Theory where it is said that a surprising event has higher information content than an expected event (Shannon, 1948)[7].

5.4.2 TF–IDF

The most popular way to weight the elements in the matrix is usually composed of three components:

$$f_{ij} = TF_{ij} \cdot DF_i \cdot S_j$$

,according to Robertson & Spärck Jones, 1997. [48] where TF_{ij} is some function of the frequency of term i in document j, DF_i is some function of the number of documents term i occurs in (DF for *document frequency*), and S_j is a normalizing factor, usually dependent on the length of document(s for scaling). The first component indicates how important word i is for document j, because the more often a term occurs in a document, the more likely it is to be important for identifying the document. DF_i is the discriminatory component– if the term appears in many documents, it should not be considered important. DF is usually computed as:

$$IDF = log\frac{D}{DF_i}$$

, where D is usually usually the total number of documents in the whole corpus. The third component Sj is normally a function of the length of document j, and is based on the idea that a term that occurs the same number of times in a short and in a long document should be more important for the short one.(Singhal, Salton, Mitra, & Buckley, 1996)[49].

5.4.3 PMI

An alternative to TF–IDF is Pointwise Mutual Information (PMI) (Church & Hanks, 1989)[9], which works well for both word–context matrices (Pantel & Lin, 2002a)[40] and term– document matrices (Pantel & Lin, 2002b).
We will explain now PMI on Information Theory and present how it is applied on the co–occurrence matrices. The Pointwise Mutual Information (PMI) between two words, and is defined as follows (Church & Hanks, 1989):

$$PMI(word_1, word_2) = log_2(\frac{p(word_1 \& word_2)}{p(word_1)p(word_2)})$$

Here, $p(word_1 \& word_2)$ is the probability that $word_1$ and $word_2$ appear together in some context, $p(word_1)$ is the probability of $word_1$, and similarly for $word_2$. The ratio between $p(word_1 \& word_2)$ and $p(word_1)p(word_2)$ product is a measure of the degree of statistical dependence between these words.
If we observe a word–context frequency matrix with n_r rows and n_c columns, where the i– th row (word vector) is represented as f_{i*}, j–th column(document–vector) as f_{*j} and frequency of word i in document j as f_{ij}, we can present how frequencies are weighted for each element of the matrix:

$$p_{ij} = \frac{f_{ij}}{\sum_{i=1}^{n_r} \sum_{j=1}^{n_c} f_{ij}}$$
$$p_{i*} = \frac{\sum_{j=1}^{n_c} f_{ij}}{\sum_{i=1}^{n_r} \sum_{j=1}^{n_c} f_{ij}}$$
$$p_{*j} = \frac{\sum_{i=1}^{n_r} f_{ij}}{\sum_{i=1}^{n_r} \sum_{j=1}^{n_c} f_{ij}}$$
$$PMI_{ij} = log_2(\frac{p_{ij}}{p_{i*}p_{*j}})$$

In this definition, p_{ij} is the estimated probability that the word w_i occurs in the context c_j, p_{i*} is the estimated probability of the word w_i, and p_{*j} is the estimated probability of the context c_j. If w_i and c_j are statistically independent, then p_{i*} p_{*j} equals p_{ij} (by the definition of independence), and thus PMI_{ij} is zero (since $\log(1) = 0$). The product $p_{i*}p_{*j}$ is what we would expect for p_{ij} if w_i occurs in c_j by pure random chance.

On the other hand, if there is an interesting semantic relation between w_i and c_j, then we should expect p_{ij} to be larger than it would be if w_i and c_j were completely indepedent. In that case we should find that $p_{ij} > p_{i*}p_{*j}$, and thus PMI_{ij} is positive. If the word w_i is unrelated to the context c_j, we may find that PMI_{ij} is negative.

A variation of PMI is Positive PMI (PPMI), in which all PMI values that are less than zero are replaced with zero (Niwa & Nitta, 1994)[39]. Bullinaria and Levy (2007)[5] demonstrated that PPMI performs better than a wide variety of other weighting approaches when measuring semantic similarity with word–context matrices. PPMI is designed to give a high value to x_{ij} when there is an interesting semantic relation between w_i and c_j ; otherwise, x_{ij} should have a value of zero, indicating that the occurrence of w_i in c_j is uninformative.

PMI is biased towards infrequent events. Consider the case where w_i and c_j are statistically dependent: Then $p_{ij} = p_{i*} = p_{*j}$. Hence x_{ij} becomes $log(\frac{1}{p_{i*}})$ and PMI increases as the probability of word w_i decreases.

PMI was tested in many experiments, one of them being the TOEFL synonym test, where its performance was was found to be better than LSA: 73.75% over LSA's 64.4% (36% without the use of SVD) (Turney, 2001)[51].

5.5 Calculating similarity

When we have the co–occurrence matrix the question is how to use it? How to extract useful information from it?

As previously mentioned in Chapter1.2 an n–dimensional vector identifies a location in an n–dimensional space. However, the vector in isolation does not contain any meaningful information. It is the distance or proximity from other vector representations of word meanings that establishes a word's meaning. In the Vector Space Model of word's meaning, *similarity of words is a geometrical proximity between their vectors*. Thus, the point of the context vectors is that they allow us to define (distributional, semantic) similarity between words in terms of vector similarity.

There are many ways to compute the similarity between vectors(raw or weighted). Large similarity produces a small distance in vector space, and opposite. The arguably simplest vector similarity metric is the scalar (or dot) product between two vectors $\vec{x}$ and $\vec{y}$, computed as:

$$sim(\vec{x}, \vec{y}) = x \cdot y = x_1 y_1 + x_1 y_1 + .. + x_n y_n \tag{5.4}$$

Another simple metric is the Euclidian distance, which is measured as:

$$dist_E(\vec{x}, \vec{y}) = \sqrt{\sum_{k=1}^{n} (x_i - y_i)^2} \tag{5.5}$$

These measures are not ideal to use for word–space algorithms: scalar product favors frequent words (i.e. words with many and large co–occurrence counts will end up being too similar to most other words), while Euclidian metrics have the opposite problem (frequent words will end up being too far from the other words)[57] .

A convenient way to compute normalized vector similarity is to calculate the cosine of the angles between two vectors $\vec{x}$ and $\vec{y}$, defined as:

$$sim_{COS}(\vec{x}, \vec{y}) = \frac{\sum_{k=1}^{n} x_i y_i}{\sqrt{\sum_{k=1}^{n} x_i^2} \sqrt{\sum_{k=1}^{n} y_i^2}} \tag{5.6}$$

Many other similarity measures have been proposed in both IR (Jones & Furnas, 1987)[25] and lexical semantics circles (Dagan, Lee, & Pereira, 1999)[11]. It is commonly said in IR that, properly normalized, the difference in retrieval performance using different measures is insignificant (van Rijsbergen, 1979)[54]. Often the vectors are normalized in some way (e.g., unit length or unit probability) before applying any similarity measure. Popular geometric measures of vector distance include Euclidean distance and Manhattan distance. Distance measures from information theory include Hellinger, Bhattacharya, and Kullback–Leibler. Bullinaria and Levy (2007)[5] compared these five distance measures and the cosine similarity measure on four different tasks involving word similarity. Overall, the best measure was cosine. Other popular measures are the Dice and Jaccard coefficients (Manning et al., 2008)[36]. Determining the most appropriate similarity measure is dependent on the similarity task, the sparsity of the statistics and the frequency distribution of the elements being compared.
As you may have noticed– the cosine measure corresponds to taking the scalar product of the vectors and then dividing by their norms. The cosine measure is the most frequently used similarity metric in word–space research, and the one we will use as well in this thesis. It is attractive because it provides a fixed measure of similarity and it ranges from 1 for identical vectors, over 0 for orthogonal vectors, to –1 for vectors pointing in the opposite directions.

6. Experiments

In the previous chapter we have outlined all the important steps in the process of constructing Vector Space Models, and listed all major approaches that we thought to be important for the task of Word Sense Disambiguation. There are certainly many other approaches that are employing VSM in some CL or NLP task, but because their domain of application is not relevant for the task of interest they where not mentioned here. This chapter is devoted to describing the full methodology behind the experiments performed in this thesis as well as how this methodology stems from previous approaches in VSM, in all the relevant phases of the process. First we will describe the resource that WSD was performed on, Prague Dependency Treebank, which will be followed with data preprocessing section. In the section after we will describe all the models implemented for the WSD task. After that the evaluation rationale will be presented. All experiments were performed on the principle: one model– one experiment. In the final sections of this chapter results of experiments with the baseline random guessing scores will be presented.

6.1 The resource

Prague Dependency Treebank (PDT) is project that stemmed from work of a group of Czech linguists (Institute of Formal and Applied Linguistics[1], Institute of Theoretical and Computational Linguistics[2]) from Charles University[3] in Prague and Masaryk University[4] in Brno. PDT was inspired by the research resulting from the Penn Treebank[5] project.

PDT was generated in two major phases. In the first phase (1996–2000), the morphological and syntactic analytic layers of annotation have been completed along with the preview of tectogrammatical layer annotation available as PDT 1.0. During the second phase (2000–2004), the tectogrammatical layer of annotation was completed and PDT 2.0 was done.
The structure of Prague Dependency Treebank (PDT) consists of three layers:

- morphological layer (lowest)– full morphological annotation
- analytic layer (middle) – superficial (surface) syntactic annotation using dependency treebank; level conceptually close to the syntactic annotation used in the Penn Treebank
- tectogrammatical layer (highest) – level of linguistic meaning

In our experiments we were using as a resource PDT1.0 version, instead of PDT2.0. Frequencies for each layer in PDT1.0. are given in the table below:

[1] http://ufal.ms.mff.cuni.cz/
[2] http://utkl.ff.cuni.cz/
[3] http://www.cuni.cz/
[4] http://www.muni.cz/
[5] http://www.cis.upenn.edu/ treebank/

	# of tokens	# of sentences
morphological (total)	1,725,242	111,175
syntactic–analytic (total)	1,507,333	98,263
tectogrammatical (portion)	3,490	203
morphological and syntactic–analytic	1,255,590	81,614

Table 6.1: Frequencies of tags for every layer in PDT1.0.

Text Sources

The text material that was annotated in PDT contains samples from the following sources:

- Lidové noviny[6] (daily newspapers), 1991, 1994, 1995
- Mladá fronta Dnes[7] (daily newspapers), 1992
- Ceskomoravský Profit (business weekly), 1994
- Vesmír (scientific magazine), Academia Publishers, 1992, 1993

6.1.1 Polysemous words

Czech language is the typical representative of inflectionally rich free–word–order language[8], which surely doesn't have a favorable influence on the number of polysemous words in Czech language. From the training set that represents approximately 9/10 of the whole PDT1.0 we present in the table below the number of words that have more than one meaning. In the left column is given the number of meanings, and in the right column the number of words that take that number of meanings. For instance, the eight row informs us that there are two words that take 8 different meanings in the training set. Another interesting ratio to be taken into account is the number

6.1.2 Why PDT1 instead of PDT2?

The question from the title of the section is indeed a perfectly valid and reasonable question indeed, for any type of research– why use an older version of a resource, when there is a newer, more complete version of the same resource?

To answer this question we must go back to the task that is in focus of this research. For the WSD task the important feature of the text on which the system is trained and tested on is that every different meaning of a word is marked orthographically different. This information that could be extracted both

[6] http://www.lidovenoviny.cz/
[7] http://www.idnes.cz/
[8] http://www.czech–language.cz/index.php

# of meanings	# of types
2	329
3	37
4	20
5	6
6	4
7	4
8	5
9	3
10	3
	Total =411

Table 6.2: Number of types with multiple meanings in the training set

from PDT1.0 and PDT2.0, therefore for the WSD task one resource is as much sufficiant as the other one. The only difference between the two treebanks is that PDT2.0 contains additionally the tectogramaticall layer, that was not used in this research.

When building(training) Vector Space Models, a researcher might want to include POS information with every tag. This was not done in our experiments but it was always an option supported by the resource, because like previously mentioned, PDT1.0 is fully annotated on the syntactic level.

On top of these reasons, another incentive that made us decide for PDT1.0 is that it essencially represents a light–wight version of PDT2.0, encoded in Standard Generalized Markup Language (SGML[9]), which made it easier to handle during the preprocessing stage.

6.2 Preprocessing

The preprocessing phase involves several several sub–stages. As we have described theoretical approaches to these steps in length in previous chapters, we have decided to explain all the preprocessing steps performed in experiments in this chapter. Preprocessing involves following sequential steps: text extraction from PDT files along with relevant annotation, text normalization, stemming and filtering.

Text extraction

Text used during training and testing phase of VSM was extracted from PDT files and put into a single file. In order to make the implementation applicable for other resources other than PDT we have made a design decision that the implementation uses this single file as its input. Apart from the single–meaning words, the system extracts multiple meaning words, and represents differently each separate meaning. In PDT1.0 if a word has multiple meanings,

[9]http://www.w3.org/MarkUp/SGML/

meaning of that word is given in a separate tag, in the following format: *word–index_number_of_meaning*. Index number increments as the word has more meanings.

Therefore, the annotation step in this case was simplified: *every multiple–meaning word was represented with its orthographic representation meaning in context*, which was simply extracted from the Treebank. An example of a sentence extracted from PDT is given below:

(6.1) Krumbachová práci pȓijmout–2 podle–2 vlastní–1 slov " v–1 hodinê dvanácté ".

Krumbach has accepted the work " in the last minute" .

Word's meaning in context was given as it was represented in PDT.

Text normalization

Czech language has spelling variants– words that are orthographically slightly different but are actually identical on every other level of analysis, for example words like *abcházská* and *abchazská*. Many foreign names have variant spellings, especially in marking the vowel length. Spelling variants should not be counted as different types so it was therefore decided to normalize all characters with diacritics marking the vowel length into characters without them.
As a measure of normalization that would reduce distributional sparsity of words all words were lowercased. Punctuation marks were also filtered from the entire corpus, so that the models could be trained on terms only.

Stemming

Stemming process was detailed in the chapter5.1, as well as how it is different from lemmatization. To re–iterate: lemmatization is more precise than stemming because it picks up lexical variants that are not orthographically similar to the stem. At the same time it is much harder to implement than a stemmer, because it requires utilization of a lexicon.

Although a good lemmatizer was built at UFAL[10], it was not used in this system. During preliminar experiments with cosine-weighted co-occurrence matrix it was observed that the improvement in precision when applying lemmatizer over stemmer is not significant. It was therefore assumed that precision would not be improved in other models experimented with as well. A stemmer built for the Czech language made at University of Neuchatel[11] was used in preprocessing phase. An example of text that was extracted, normalized and stemmed is given below:

(6.2) krumbach prak pȓijmout–2 podle–2 vlastni–1 slov " v–1 hodin dvanact ".

Krumbach has accepted the work " in the last minute" .

Stemming was left as an optional measure in order to observe whether it improves the precision of algorithm.

Word Filtering

[10]http://ufal.mff.cuni.cz/pdt/Morphology_and_Tagging/Morphology/index.html
[11]http://members.unine.ch/jacques.savoy/clef/CzechStemmerLight.txt

Two effective filtering criteria were applied (optionally) in our experiments for the purpose of reducing the sparsity of vectors and improving overall precision: POS and low frequency counts. A list of Czech stop words was compiled(Apendix A) in order to filter out words that belong to closed grammatical classes. Both types of filtering were experimented with, in order to observe how their tunning influences the overall precision of system.

6.3 Train and test sets

After extracting the text from PDT and optional preprocessing normalization performed on text, text's sentences are randomized and split into 3 separate files: $train, testDev, testFinal$. Training set holds 85% of the number of sentences in the whole set, while *testDev* holds about 5% and $testFinal$ about 10%. First test file (*testDev*) is used to evaluate during training phase, while the second ($testFinal$) is used for evaluation in the testing phase.

Below is given an overview of the number of types and tokens for each of these files, before and after preprocessing applied to them:

	train set		test devel set		final test set	
	#types	#tokens	#types	#tokens	#types	#tokens
no preprocessing	114720	1080400	10439	32105	27626	113330
merge variants	114696	1079851	10418	32033	27530	112678
stemming	62063	1050948	7699	29891	18122	105912

Table 6.3: Influence of the preprocessing method on the size of training and test sets

We can observe that stemming reduces the vocabulary size much more than merging lexical variants. This fact will reflect on the precision in the evaluation phase.

6.4 Document size

After preprocessing phase, the next is to build the term–document matrix. This step is generalized for all models we will be training. This is because the term–document matrix is a simple frequency matrix, and (optional) normalization of frequency counts or dimensionality reduction takes place in the next phase. Crucial property of this phase is the size of the document in the term–document. Previous approaches were described on the use of context, ranging from word–level to paragraph–level. Document size used in the following experiments will be:

- five sentence paragraph
- three sentence paragraph
- one sentence paragraph
- three neighboring words of preceding and succeeding context

- two neighboring words of preceding and succeeding context
- one neighboring word of preceding and succeeding context

To further clarify what exactly do last three items in the list mean, we will point out that the since the task at hand is disambiguation of polysemous words. Therefore, polysemous meanings should be in focus of every document. We have decided to conduct experiments by partioning sentences into fragments where in the center of every such fragment is a polysemous meaning, surrounded by symmetrical window length of varying size. An example is given for a one neighboring word of preceding and succeeding context. For a sentence:

(6.3) Krumbachová práci pȓijmout–2 podle–2 vlastní–1 slov " v–1 hodinê dvanácté ".

Krumbach has accepted the work " in the last minute" .

, corresponding fragments extracted from it are:

(6.4) práci pȓijmout–2 podle–2

(6.5) pȓijmout–2 podle–2 vlastní–1

(6.6) podle–2 vlastní–1 slov

(6.7) slov v–1 hodinê

Although this fragmentation increases the volume of documents we believed that centering somehow documents around polysemous encounters will help train the model better. The rationale behind that decision was that this was an assumption that this way the model will become "more sensitive" to assigning correct meanings to their rightful contexts.

Choice of document size during model training is one of the crucial features of VSM, and can very much influence the results of the model's performance, through the size of vocabulary. Size of vocabulary is influenced by preprocessing method applied as well the size of the document. Below is given a table that shows how the choice of document size influences the size of vocabulary, for the corpus that was not preprocessed.

document size	#index documents	ambiguous words #types	#tokens	other words #types	#tokens
5 sentences	15732	3513	4721	128676	1155781
3 sentences	26220	3334	4417	129618	1166559
1 sentence	78660	2116	2660	130620	1216412
3+3	231673	2116	2660	105225	1433731
2+2	231673	2116	2660	94018	1088417
1+1	231673	2116	2660	70772	700799

Table 6.4: Influence of document size on the volume of vocabulary

With the word–level document size we can observe that number of types is smaller than in sentence –level documents. This is due to existing gaps between documents in matrix with word–type document. Number of tokens can be bigger which we see in the case of 3+3 document size, because in some places, where two occurrences of ambiguous words stand next to each other, there will be an overlap which will be counted.

6.5 Normalization of the frequency counts and dimensionality reduction

Normalizing the frequency counts was done in two ways in experiments: the traditional TF–IDF weighting, and the enthropy–based weighting as prescribed in LSA. To re–iterate the formulas again:

$$f_{ij} = TF_{ij} \cdot DF_i \cdot S_j \tag{6.8}$$

,where TF_{ij} is the frequency of term i in document j, DF_i is some function of the number of documents term i occurs in (DF for *document frequency*), and S_j is a normalizing factor, usually dependent on the length of document(s for scaling). DF is usually computed as:

$$IDF = log\frac{D}{DF_i} \tag{6.9}$$

, where D is the total number of documents in the whole corpus.

Another frequency weighting performed was the entropy–based weighting as given in the formula below:

$$E_{ij} = 1 + \frac{\sum_j P_{ij} log P_{ij}}{log D} \tag{6.10}$$

where D is the total number of documents in the collection. P_{ij} is given with a formula

$$P_{ij} = \frac{TF_{ij}}{f_i} \tag{6.11}$$

where TF_{ij} is the frequency of term i in document j and f_i is the frequency of term i in the whole document collection.

Another model that was experimented with is created through the use of dimensionality reduction technique– Random Indexing. Frequencies counts of its term–document matrix were not normalized, due to the nature of its generation. Detail description of RI is given in chapter4.5. What should be restated is that:

- It builds term vectors by adding pseudo–random vectors assigned to documents in the corpus.
- It uses fixed dimensionality for both term and document vectors, which means that new data do not increase the dimensionality of the vectors.
- It uses implicit dimensionality reduction, since the fixed dimensionality is much lower than the number of contexts in the data. Producing context vectors with RI is only $O(wr)$, since the method is not reliant on the initial construction of the co-occurrence matrix.

- It is incremental, which means that the context vectors can be used for similarity computations even after just a few examples have been encountered.

More details on RI can be found in Chapter4.5.

The two approaches to weighting the co–occurrence matrix(TF–IDF and PMI) and one approach to dimensionality reduction (RI) are considered as fundamentally different. That is why the experiments are organized around these models, with other parameters like preprocessing, document size, and evaluation context size and number of word meanings threshold used in the evaluation(both explained in following sections) are changed to investigate how they influence the overall performance.

6.6 Evaluation context size

After the co-occurrence matrix is built and frequencies of its elements are normalized with some value (TF–IDF or PMI), or the matrix's dimensionality is reduced (RI) our model is ready to be tested out. This means that now for every word in the dictionary we have a vector that is in a way its representation. A unique methodology is then applied for each model in order to calculate the measure of appropriateness for a current word meaning against the context it is found in. Size of the context around the current meaning in question is experimented with and in our experiments takes values from 1 to 7, to determine the influence of symmetric context size on the overall accuracy. Evaluation context size is the number of words taken from preceding and succeeding context of the occurrence of ambiguous word. For instance, for the sentence:

(6.12) Krumbachová práci pȓijmout–2 podle–2 vlastní–1 slov " v–1 hodinê dvanácté ".

Krumbach has accepted the work " in the last minute" .

, corresponding contexts for the evaluation context size set to 1 extracted from it are:

(6.13) pȓijmout–2 : práci podle–2

(6.14) podle–2: pȓijmout–2 vlastní–1

(6.15) podle–2 slov

(6.16) slov hodinê

In the implementation level, the way the most appropriate meaning for a context is calculated in two passes:

- The test set is traversed, in order to extract all the contexts for all the polysemous words encountered in testing.
- Extracted contexts are traversed, and for every different meaning the appropriateness for that word meaning in context is calculated. The meaning that has the highest score is selected as the true meaning by the system.

Two things here are of importance: the way context vectors is built, and the way distance between context vector and meaning vector is calculated. Context vector in all cases (TF–IDF, PMI and RI) is built by superposition of all the term vectors constituting that context. If the term was not encountered during training it is skipped. If context vector does not have any terms inside it then that context is skipped during evaluation. This means a difference in size of vocabulary made in training can influence the number of contexts being evaluated, and therefore the test set size.

Distance between context and meaning vector is calculated as a cosine distance in all cases.

To repeat, cosine distance is the angle between two vectors $\vec{x}$ and $\vec{y}$, defined as:

$$sim_{COS}(\vec{x}, \vec{y}) = \frac{\sum_{k=1}^{n} x_i y_i}{\sqrt{\sum_{k=1}^{n} x_i^2}\sqrt{\sum_{k=1}^{n} y_i^2}} \tag{6.17}$$

6.7 Evaluation metrics and baselines

In this section evaluation metrics and baselines used in this research will be presented, employed for *in vitro* evaluation of WSD systems (Navigli 2009). The evaluation the WSD system as a module embedded in applications (called *in vivo* or *end-to-end* evaluation) was not conducted in this research.

All sentences extracted from the PDT and annotated to distinguish different meanings of polysemous words are divided into three sets (files): training, test development, and test final. Training takes about 9/10 of the entire set, while the rest is distributed evenly between two test sets. During the training phase the model is trained on the training set, and evaluated on the test development set. In the evaluation phase training and test development sets are merged, the model is trained on them, and evaluated on the final test set, which represents the unseen portion of data. This ensures an unbiased evaluation of every model.

Given the individual training and test sets *coverage* C is defined as the percentage of items in the test set for which the system provided a sense assignment:

$$C = \frac{\#answers\_provided}{\#total\_answers\_to\_provide} = \frac{TP + FP}{TP + FP + FN} \tag{6.18}$$

If the systems provides an answer for every test instance then $C = 1$. The *precision* P of a system is computed as the percentage of correct answers given by the system. If the model "calculates the correct meaning" properly, $\#correct\_answers\_provided$ is incremented. The total number of answers given by the system is counted as well, and then precision is calculated as:

$$P = \frac{\#correct\_answers\_provided}{\#answers\_provided} = \frac{TP}{TP + FP} \tag{6.19}$$

Precision determines how good are the answers given by the system. *Recall R* is defined as the number of correct answers given by the automatic system over the total number of answers to be given:

$$R = \frac{\#correct\_answers\_provided}{\#total\_answers\_to\_provide} = \frac{TP}{TP + FP + FN} \quad (6.20)$$

If the coverage of the system is absolute ($C = 1$) then recall and precision will be equal. From these two values (precision and recall) an F–measure is calculated according to formula:

$$F - measure = \frac{1}{\frac{\alpha}{P} + \frac{1-\alpha}{R}} \quad (6.21)$$

where α is the weight factor which in experiments performed here takes the value of 0.5, in order to give equal weights to both precision and recall.

Baseline.
As a baseline result, random guessing is calculated for every test set used in evaluation. Random baseline is the random choice of a sense from those available for each polysemous word w_i . Random precision is calculated as:

$$RandomP = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{Senses(w_i)} \quad (6.22)$$

where $Senses(w_i)$ is the number of senses that a w_i can have.

6.8 Tuning

Number of meanings that polysemous words encountered in the corpus can vary from 1 to 8 meanings. The actual number of meanings follows Zipf's distribution, where ranks are the number of meanings word can take (1 to 8) and numbers are the number of polysemous words that take that number of meanings. To illustrate this with a simple example: number of words that take 2 meanings in PDT1.0 is 329. Full table is given in chapter6.1.1. Below is given a plot to illustrate how meaning's counts and word types conform to the Zipfian distribution.

Three types of experiments for each model were performed, in order to tune each model to achieve its best performance.

6.8.1 Tuning preprocessing parameters

Each of the preprocessing measures mentioned before (lowercasing, filtering words that belong to the closed class group, stemming and merging of lexical variants in Czech language) was experimented against the different document size (5,3,1 sentence document and 3+3, 2+2, 1+1 words surrounding polysemous word) to

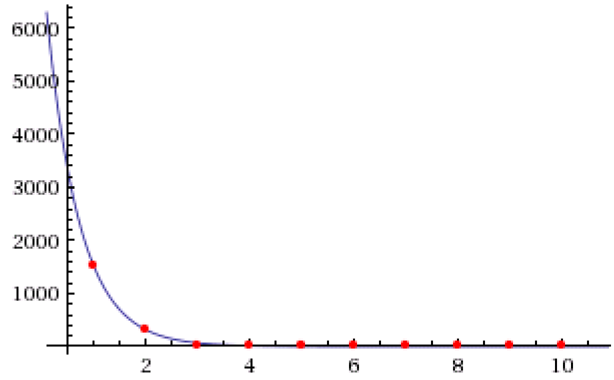


Figure 6.1: Distrubution of number of polysemous types per number of meanings

determine for which parameters the model achieves the highest accuracy. These models were evaluated only for highly polysemous words, at the fixed size of the evaluation context size. In order to able to test how models really perform we train models only on highly polysemous words, the ones that have 6 or more meanings. There are 19 polysemous words in the test development set that can take 6 to 10 meanings.

6.8.2 Tuning evaluation context size

When the best parameters for preprocessing and document size (in the term–document matrix) are discovered, the next tuning phase experiments with different sizes of Evaluation Context against these best paremeters. There are two motivations for this sequence of tuning experiments: first, the preprocessing and building the matrix come before matrix frequency normalization and evaluation. Second, after the best parameters in the preprocessing stage are found, we can observe how this (best) model behaves with different sizes of context (from which the evaluation context vector is constructed).

6.8.3 Evaluation on polysemous words of different level

Words whose orthographic representation can take on a large number of meanings can be considered as ”highly polysemous", while words that can take on a small number of meanings(like two or three) can be regarded as ”not highly polysemous". As explained, numbers of occurrences of polysemous words conform to the Zipfian distribution, which means that there is only a small number of highly polysemous words. Therefore, when experimenting with such words test set is not overly large, but if other, less polysemous words are included, the number of test set instances rise. The purpose of this experiment is to investigate how do models handle more and less polysemous words. Ranges from 2 to 9 meanings per word are taken into consideration.

6.9 TF–IDF experiments

As previously mentioned experiments are organized around 3 different different Vector Space Models, based on the weighting of the matrix elements or dimensionality reduction technique. Matrix reduction technique used for this model was TF–IDF (described in Chapter 6.5). For all models experiments were performed on every possible parameter, grouped into under the phase under which they were used:

- Tuning preprocessing parameters
- Tuning evaluation context size
- Evaluation on polysemous words of different level

These rounds are ordered sequentially in order to ensure best possible tuning for the model. Final step is evaluation on words with different number of meanings attached to them.

6.9.1 Tuning preprocessing parameters

First, no preprocessing method was applied on the train and test set. The size of the document in the term–document co- ocurrence matrix was changed to observe the influence on the precision. A threshold number of meanings was set to 6, which means that other polysemous words that had less than 6 meanings encountered in training were dicareded for this test. Also an evaluation context size was fixed to 3. Document sizes that were experimented with are: 5 sentence document, 3 sentence document, 1 sentence document, and 3+3, 2+2, 1+1 symmetric word windows around polysemous word. Random precision was calculated as well to serve as a baseline. Best results in all the tables are bolded. Precision (in %) for every document size is presented in the tables below. First a sentence–level document size was experimented with.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
NO	93.65	100	94.23	100	93.84	100	29.38
LOWCASE	97.56	100	95.71	100	95.28	100	7.3
STEM	99.48	100	98.95	100	99.48	100	14.12
MERGE	99.48	100	98.95	100	99.48	100	14.12

Table 6.5: Precision and Recall of TF–IDF model, no preprocessing, sentence level document size

As explained in the previous section evaluation was performed in two phases. Only results from the evaluation phase are displayed in the tables. Random prediction from the training phase was 8.1%. Total number of final test set instances was **17845**, however because only words with 6 and more meanings were taken into consideration test set size was **952**, while even **16882** occurrences of polysemous words was discarded.

Then word–level document size was then observed to see how it performs.

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
NO	90.43	93.61	**90.90**	**94.65**	85	93.22	4.51

Table 6.6: Precision and Recall of TF–IDF model, no preprocessing, word level document size

Next preprocessing technique that was experimented with was lowercasing. Total number of final test set instances was **18383**, however the test set size was **9636**, while **8747** occurrences of polysemous words was discarded for having less than 6 meanings attached to them. Tables are given below.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
LOWCASE	97.08	100	96.45	100	97.09	100	50

Table 6.7: Precision and Recall of TF–IDF model, lowercased, sentence level document size

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
LOWCASE	**85.44**	**86.36**	85.04	86.44	83.33	86.19	4.9

Table 6.8: Precision and Recall of TF–IDF model, lowercased, word level document size

In the next experiment we will investigate how filtering out words based on their belonging to a group of closed-class words and lowercasing influence the precision. Total number of final test set instances was **17178**, however the test set size was **912**, while **16261** occurrences of polysemous words was discarded for having less than 6 meanings attached to them. Terms which appear in too many documents (e.g., stopwords, very frequent terms) receive a low weight, while uncommon terms which appear in few documents receive a high weight. This makes sense since too common terms (e.g., "a", "the", "of", etc) are not very useful for distinguishing a relevant document from a non-relevant one. The two extremes are not recommended in routine retrieval work.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
CLOSED GROUP	90.07	93.8	87.05	91.35	**94.58**	**92.65**	4.3

Table 6.9: Precision and Recall of TF–IDF model, filtering words that belong closed group class, sentence level document size

Next preprocessing measure that was tested was stemming plus merging lexical variants. Seen that merging of lexical variants does not reduce the vocabulary significantly, we have decided to join it with stemming, and observe the

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
CLOSED GROUP	**91.16**	**94.58**	90.72	92.59	87.5	94.34	4.3

Table 6.10: Precision and Recall of TF–IDF model, filtering words that belong closed group class, word level document size

results. Total number of final test set instances was **17301** , the test set size was **8232**, while **9069** occurrences of polysemous words was discarded for having less than 6 meanings attached to them. It should be noted that this was the largest test set.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
STEM+MERGE	40.08	88.28	83.51	85.41	**83.86**	**85.62**	9.5

Table 6.11: Precision and Recall of TF–IDF model, stemming plus merging Czech lexical variants, sentence level document size

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
STEM+MERGE	**83.05**	**85.46**	82.48	85.37	78.28	84.72	9.5

Table 6.12: Precision and Recall of TF–IDF model, stemming plus merging Czech lexical variants, word level document size

Finally, preprocessing measures that gave best results were combined and tested to observe whether this kind of preprocessing outperforms the most successful preprocessing method. Total number of final test set instances was **17120**, the test set size was **7950**, while **9170** occurrences of polysemous words was discarded for having less than 6 meanings attached to them.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
CLOSED CLASS+ STEM+MERGE	33.24	69.15	82.24	84.63	**82.5**	**84.82**	4.3

Table 6.13: Precision and Recall of TF–IDF model, combined preprocessing methods, sentence level document size

Discussion

Although results achieved without any preprocessing whatsoever achieve good results one has to bear in mind that the size of the test set is not that large and that more than 60% of test set data was discarded as unsuitable for this experiment for having to few meanings attached to them. Best achieved result

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
CLOSED CLASS+ STEM+MERGE	**81.05**	**84.54**	80	84.36	75.53	83.55	4.3

Table 6.14: Precision and Recall of TF–IDF model, combined preprocessing methods, word level document size

without preprocessing was accomplished for 1 sentence document size and 2+2 word window for sentence level and word level document, respectively.

Lowercasing produced a somewhat smaller vocabulary over the entire corpus, which resulted in merging several word types and producing more highly polysemous words with which then in return occurred in more contexts thus resulting in the larger test set. Still, the accuracy of the algorithm was at a very high level.

Filtering words that belong to closed class of words gave the best results out of all preprocessing approaches. Precision and recall were at a very high level, thought at not so high level of size of the test set. Moreover, it reduced the word space, which considerably influenced the time of computation. If it was to be compared with the test of the set that was not preprocessed, it can be observed that testing in this case was performed 5 times faster. This is viewable from the logs (see User Manual).

Stemming and merging variants produced good results as well, at a high level of test set size. All models perform outstanding compared to random prediction.

Conclusion.

It can be observed that filtering out words that belong to closed group produced best results. Combination with filtering closed group words produced somewhat weaker results. This can be explained that after a lot of merging and cleaning, the "word space" was for this test data (PDT) ”too clean", thus preventing the model to perform correct discrimination between all meanings of an ambiguous word in a given context.

Another thing which can be observed for all the values of parameters is the document size which produced best results. For TF–IDF model a golden middle seems to yield best results, somewhere between 1 sentence document, and 3+3 symmetric window around ambiguous word.

6.9.2 Tuning evaluation context size

Evaluation context size is the size of symmetric window of context which is taken into consideration when the polysemous word is encountered during testing. Words that are found within that context window are used to form a context vector. This context vector’s distance is measured against every vector of each of the meanings for the polysemous word in question. Sizes of context vector experimented with range from 1 to 7. In order to have a larger test set, number of meanings that a word can take was lowered to 5 instead of 6 like in the preprocessing experiments. Preprocessing performed was filtering out closed class words, and document size was set to 1 sentence. Results are given in the table below.

	1+1	2+2	3+3	4+4	5+5	6+6	7+7
Precision	96.03	95.58	95.48	95.70	95.72	95.67	95.52
Recall	98.67	98.31	98.16	98.13	98.09	98.07	98.05

Table 6.15: Precision and Recall of TF–IDF model, closed class filter, document size 1, word level document size

Both precision and recall have very similar result for all evaluation context sizes. Variance in both sets is very low, hence there is not much difference in which context size will be used for final evaluation of all occurrences of polysemous words.

6.9.3 Evaluation on polysemous words of different level

So far we have been training the model on highly polysemous words, the ones that have 6 or more meanings attached to them. Random prediction on these highly polysemous words is very low, therefore they present a better, more unbiased test set than for instance words that have only 2 meanings, and for which random accuracy is 50%. We will now observe the accuracy of the model set with best parameters to see how it performs on different sets of words with the same number of meanings. This means that for instance, in the first column are results of experiments performed on words with 2 meanings attached to them.

	2	3	4	5	6	7
Precision	94.04	97.19	83.97	97.28	59.3	97.424
Recall	99.80	99.82	99.26	99.70	79.16	98.99
Random prediction	50	33.33	25	20	16.66	14.28

Table 6.16: Precision and Recall of TF–IDF model, closed class filter, document size 1, evaluation context size 3+3 measured against words with different number of meanings

Conclusion We can see that for every set the model performs well, although some results are somewhat surprising , for instance precision for words with just two meanings is lower than precisions of all words with more meanings(6 meaning words excluded). This could be explained with the fact that the frequency of appearance of two meaning words is higher than for other cases (it had the highest number of appearance in test set –2540), thus making it harder to discriminate two meanings. Other explanation is that two–meaning words can appear in each other's contexts. As far as the lowest result is concerned (6 meaning words), the explanation lies in the number of test set instances: there was only 32 of them (tp=19.0 fp=13.0). Low number makes this result statistically insignificant.

Overall, we can say that TF–IFD model performed well at this task.

6.10 PMI experiments

Experiments with PMI model are like the ones with TF–IDF, grouped in three rounds in order to tune the model for the best performance.

- Tuning preprocessing parameters
- Tuning evaluation context size
- Evaluation on polysemous words of different level

6.10.1 Tuning preprocessing parameters

Being that preprocessing methods were thoroughly examined in section about tuning the preprocessing parameters for TF–IDF we have a general idea which values for document sizes and preprocessing methods give best values. We will thus be concentrating on a smaller set of preprocessing methods. Results will be given in summed tables for all preprocessing methods, measured against different document sizes. Other parameters are fixed like in TF–IDF tuning experiments: evaluation context size is set to 3, and threshold number of meanings set to 6. Below are given tables for sentence and document level.

preprocessing	5 sentences		3 sentences		1 sentence		random
	P	R	P	R	P	R	
NO	89.81	92.53	90.0	92.23	**90.57**	**93.34**	4.5
CLOSED CLASS	89.65	92.54	89.71	92.21	**90.43**	**91.88**	4.3
STEM+MERGE	34.71	71.04	83.51	85.41	**83.86**	**85.62**	9.6

Table 6.17: Precision and Recall of PMI model, various preprocessing methods, sentence level document size

preprocessing	3+3 words		2+2 words		1+1 word		random
	P	R	P	R	P	R	
NO	90.43	93.61	**90.61**	**93.61**	85.07	93.22	4.5
CLOSED CLASS	**91.16**	**94.58**	90.72	94.53	87.5	94.34	4.3
STEM+MERGE	83.05	85.46	85.37	83.9	**85.67**	**94.74**	9.6

Table 6.18: Precision and Recall of PMI model, various preprocessing methods, word level document size

Total number of final test set instances without preprocessing was **17845**, the test set size was **952** , while **16893** occurrences of polysemous words was discarded for having less than 6 meanings attached to them.

Total number of final test set instances for word filtering was **17220**, the test set size was **928**, while **16292** occurrences of polysemous words was discarded for having less than 6 meanings attached to them.

Total number of final test set instances for stemmed set was **16761**, the test set size was **7897**, while **8864** occurrences of polysemous words was discarded for having less than 6 meanings attached to them.

7. Implementation

Software used in all the experiments in this thesis is developed under the name PDT Word Sense Disambiguator. Entire system was implemented in Java programming language. Two third–party libraries were used:

1. lucene[1]: for the purpose of text indexing, released under the Apache Software License and

2. semantic vectors[2]: for the purpose of constructing random vectors, released under the BSD 2-Clause License [3]

Classes are separated into following packages: preprocessing, vectorModels, utils (contains data types, string manipulation classes, vector operation classes), evaluation and experiments.

7.1 Data flow

In this section will be presented an outline of sequential steps taken by the system in order to train the model, and then evaluate it on the test set. The options mentioned here are dicussed in length in previous chapters, therefore I will outline here only what is relevant for the data flow.There are 4 major stages that subsume a number of smaller, potential steps:

Obtaining and dividing the data
First step here is to extract the text from PDT1.0(which is in a form of fully annotated SGML text). Different meanings of ambiguous word are annotated differently, so this information is kept, while all other words are extracted in their normal form. This part is vital because the model needs to be trained on different meanings in order to be able to differentiate them during testing. When the text is obtained it is divided into three sets: training, testing for development and final testing. Sentences from original data set are randomized before division. Training takes about 9/10 of the entire data set, while 1/10 is evenly distributed to remaining two sets. Evaluation is performed twice: during training, the model is trained on training set, and tested to development test set. During testing, model is trained on train+testDev set and evaluated on final test set, which represents an unseen portion of data. Files that are trained on and later tested on are run through the indexer. Every vector used in experiments is built straight from the indexer.

Preprocessing
There are 4 preproccessing options (all optional): lowercasing, stemming, filtering words that belong to closed class, and merging Czech lexical variants. Lowercasing and word filtering take place in the indexer: lowercasing is just a matter of option, while for word filtering a stop word list needs

[1]http://lucene.apache.org/java/docs/index.html
[2]http://code.google.com/p/semanticvectors/
[3]http://www.opensource.org/licenses/bsd-license.php

to be passed. This list is to be found in the Apendix 1. Stemming and merging of czech variants are performed by the program directly on files. If one of these two options is used, it is applied before indexing.

Building the matrix
After preprocessing indexer builds co-occurrence matrix (term–document). Documents passed to the indexer are determined by the system, and there are two options that can be set: number of sentences in one index document, and size of the context window found around ambiguous word. Any positive number can be set on either option, there is just a rule not to set number of sentences to a value larger than 1 when you want to build word–level documents. Choice of document size determines the "sensitivity" of the model, as is dicussed in Experiments chapter. During this phase all meanings of ambiguous word are saved in the hash table. Elements of the hash table are items whose keys are ambiguous words while the values attached to the keys are lists which keep all the meanings of ambiguous word. This hash table is passed to evaluator during evaluation.

Normalizing the matrix: frequency weighting or dimensionality reduction
Although in case of frequency weighting this step is performed during evaluation, it is more natural to come before it, therefore I will describe it here. Frequency weighting is performed on elements of term–document matrix, and which weight scheme is applied depends on the model (either TF–IDF or PMI). Weighting is however not applied to RI model, due to its nature (see chapter on Random Indexing). RI model is constructed with the usage of semantic vectors package. After this step, the model is considered to be trained, and is ready for the evaluation.

Evaluation
Evaluation is performed in several steps. First the entire test set is passed in order to extract all the ambiguous words and the contexts they are found in. For some reason in PDT1 some words are labeled as ambiguous although they have only one meaning. These kinds of occurrences are inserted in previously mentioned hash table during matrix construction phase. If a an ambiguous word encountered in testing is determined to have only one meaning in training, it is not put into the test set. When test set instances are extracted, model compares distances between a test set instance of ambigous word and all other meanings to the context vector. Whichever meaning is the closest by cosine distance from the context vector is predicted to be the "correct meaning". Term vectors are retrieved at this point, and in the case of TFIDF and PMI they are weighted on the spot. RI were created during indexing and stored in *termvectors.bin* file.

8. User manual

In this section it will be explained how to use the software developed for the purpose of the experiments. First section describes the parameters

8.1 Input parameters

System is accessed through a single class to which command line parameters are passed in the form:

[-argumentType argumentValue]*

Class to which the arguments should be passed is called *Experiment*.

All arguments are optional. All arguments have their default values. Class that parses arguments is called *Arguments*. An overview of all input arguments is given in the table below.

index	name	data type	default value	used for
1	lowercase	String	"n"	preprocessing
2	stopWordsRemoval	String	"n"	preprocessing
3	stemming	String	"n"	preprocessing
4	mergeLexicalVariants	String	"n"	preprocessing
5	inputFilePath	String	pdt1_cleaned	getting data
6	numberOfSentencesInLuceneDoc	Integer	1	document size
7	numberOfWordsInDocument	Integer	-1	document size
8	matrixType	Integer	0	{0,1,2}={TFIDF,PMI,RI}
9	upBoarderForNumberOfMeanings	Integer	6	evaluation
10	evaluationContextWindowSize	Integer	3	evaluation

Table 8.1: Overview of system's input arguments

If someone wishes to perform WSD on another dataset, they should specify the path to the file with the $-inputFilePath$ argument. However, it should be noted that occurrences of polysemous words should be annotated so that different meanings could be differentiated by the system. Content of the file passed to the system is split into three separate files: train, testDev, testFinal. TestFinal contains test test sentences.

8.2 Logs

For every experiment a log is made in the log folder, containing all relevant statistics related to the experiment. Log name is constructed from abbreviations of all values of experiment's phases, for instance: $PMI\_STOP{+}MERGE\_1s\_7w\_6m$ means that it is a log of PMI model, where in preprocessing filtering out closed class words and merging lexical variants was performed, document size in term–doc matrix is 1 sentence, 7+7 evaluation window context was used to construct context vectors, and only words with 6 meanings were investigated.

Summary

The primary task of this Master's thesis was to perform Word Sense Disambiguation on the Prague Dependency Treebank dataset. Through parameter tunning and comparing three different Vector Space Models, results comparable to some of the state–of–the–art approaches were achieved (on a different data set). Findings of the thesis that stem from model comparing are that the models that do not perform any matrix reduction operation on the co–occurrence matrix are the ones giving better results. Computation time for non-reducing models is also better.

Implementation of the system was made so that through passing of parameters through command line anyone can perform experiments without the need to look into the source code. Also, a data set can be easily switched (for the system to be trained on and evaluated) also through passing the command line argument containing the location of the file that holds the data set.

Bibliography

[1] *Proceedings of Senseval-3: Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text*, Barcelona, Spain, 2004.

[2] Kilgarriff Adam. I don't believe in word senses. *CoRR*, cmp-lg/9712006, 1997.

[3] Kilgarriff Adam and Palmer Martha. Introduction to the special issue on senseval. *Computers and the Humanities*, 34(1–2):1–13, 2000.

[4] Jing Bai, Dawei Song, Peter Bruza, Jian-Yun Nie, and Guihong Cao. Query expansion using term relationships in language models for information retrieval. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 688–695, New York, NY, USA, 2005. ACM Press.

[5] J.A. Bullinaria and J.P. Levy. Extracting semantic representations from word co-occurrence statistics: A computational study. *Behavior Research Methods*, (3):510, 2007.

[6] Osgood C. The nature and measurement of meaning. *Psychological Bulletin*, 49:197–237, 1952.

[7] Shannon C. A mathematical theory of communication. 27:379–423, 623–656, 1948.

[8] Ruhl Charles. *On Monosemy: A Study in Linguistic Semantics*. State University of New York Press, Albany, 1989.

[9] Kenneth W. Church and Patrick Hanks. Word association norms, mutual information, and lexicography. In *27th Meeting of the Association for Computational Linguistics*, pages 76–83, Vancouver, B.C, 1989.

[10] Lin D. Automatic retrieval and clustering of similar words. In *Proceedings of the 17th international conference on Computational linguistics*, pages 768–774, 1998.

[11] Ido Dagan, Lillian Lee, and Fernando C. N. Pereira. Similarity-based models of word cooccurrence probabilities. *Machine Learning*, 34(1-3):43–69, 1999.

[12] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41:391–407, 1990.

[13] Bingham E. and Mannila H. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2001)*, pages 245–250. Association for Computing Machinery (ACM), 2001.

[14] GW Furnas, S. Deerwester, ST Dumais, TK Landauer, RA Harshman, LA Streeter, and KE Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. pages 465–480, 1988.

[15] Salton G., Wong A., and Yang C.S. A vector space model for automatic indexing. *Communications of the ACM*, 18:613–620, 1975.

[16] D. Gentner. Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7:155–170, 1983.

[17] Zipf George Kingsley. *Human Behaviour and the Principle of Least Effort:An introduction to human ecology.* Cambridge, MA: Addison-Wesley, 1949.

[18] G. Golub and W. Kahan. Calculating the singular values and pseudo-inverse of a matrix. 1965.

[19] Rubenstein H. and Goodenough John B. Contextual correlates of synonymy. *Commun. ACM*, 8(10):627–633, 1965.

[20] Schütze H. and Pedersen J. A vector model for syntagmatic and paradigmatic relatedness. In *In Making Sense of Words: Proceedings of the Conference*, pages 104–113, Oxford, England, 1993.

[21] Schütze H. and Pedersen J. Information retrieval based on word senses. In *In Proceedings of the 4th Annual Symposium on Document Analysis and 145 Information Retrieval*, pages 161–175, 1995.

[22] Schütze Hinrich. Dimensions of meaning. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, Supercomputing'92*, pages 787–796. IEEE Computer Society Press, 1992.

[23] Schütze Hinrich. Word space. In *Proceedings of the 1993 Conference on Advances in Neural Information Processing Systems, NIPS'93*, pages 895–902, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[24] Schütze Hinrich. *Automatic word sense discrimination*, volume 24. Computational Linguistics, 1998.

[25] W. P. Jones and G. W. Furnas. Pictures of relevance: A geometric analysis of similarity measures. *Journal of the American society for information science*, 38(6):420 –442, 1987.

[26] Erk K. and Padó S. A structured vector space model for word meaning in context. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing (EMNLP-08)*, pages 897–906, 2008.

[27] P. Kanerva. *Sparse distributied memory.* MIT Press, Cambridge, MA, 1988.

[28] J. Karlgren, M. Sahlgren, and R. Cöster. Principled query processing. In *Working notes of the 6th workshop of the Cross-Language Evaluation Forum, CLEF'05, Vienna, Austria*, pages 245–250. Association for Computing Machinery (ACM), 2005.

[29] S. Katz. Distribution of content words and phrases in text and language modelling. *Natural Language Engineering*, 2(1):15–59.

[30] T. K. Landauer and S. T. Dumais. A solution to Plato's problem: the Latent Semantic Analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[31] Victor Lavrenko and W. Bruce Croft. Relevance-based language models. In *Proceedings of SIGIR*, pages 120–127, 2001.

[32] Claudia Leacock, Geoffrey Towell, and Ellen Voorhees. Towards building contextual representations of word senses using statistical models. In *Corpus Processing for Lexical Acquisition*, pages 97–113. MIT Press, 1993.

[33] Burgess C. Lund, K. and R. A. Atchley. Semantic and associative priming in highdimensional semantic space. In *Proceedings of the 17th Annual Conference of the Cognitive Science Society*, pages 660–665, 1995.

[34] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instrumentation, and Computers*, 28:203–208, 1996.

[35] C. Manning and H. Schütze. *Foundations of statistical natural language processing.* MIT Press, Cambridge, MA, 1999.

[36] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, 2008.

[37] Tom M. Mitchel, Svetlana V. Shinkareva, Andrew Carlson, Kai-Min Chang, Vicente L. Malave, Robert A. Mason, and Marcel Adam Just. Predicting human brain activity associated with the meanings of nouns. *Science*, 320:1191–1195, 2008.

[38] Shashi Narayan, Srdjan Prodanovic, Zoë Bogart, and Mohammad Fazleh Elahi. Population and enrichment of event ontology using twitter. In *First Workshop of Semantic Personalized Information Management, LREC, Malta, 2010*, May 2010.

[39] Yoshiki Niwa and Yoshihiko Nitta. Co-occurrence vectors from corpora vs. distance vectors from dictionaries. In *Proceedings of the 15th International Conference On Computational Linguistics*, pages 304–309, Kyoto, Japan, 1994.

[40] Patrick Pantel and Dekang Lin. Discovering word senses from text. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 613–619, New York, NY, USA, 2002. ACM.

[41] C. Papadimitriou, P. Raghavan, H. Tamaki, and S. Vempala. Latent semantic indexing: A probabilistic analysis. In *In Proceedings of the 17th ACM Symposium on the Principles of Database Systems*, pages 159–168. ACM Press, 1998.

[42] Edmonds Philip. *Lexical disambiguation. The Elsevier Encyclopedia of Language and Linguistics.* Oxford: Elsevier, 2005.

[43] Edmonds Philip and Scott Cotton. Senseval-2: Overview. In *Proceedings of Senseval-2: Second International Workshop on Evaluating Word Sense Disambiguation Systems*, pages 1–5, Toulouse, France.

[44] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[45] Gallant S. A practical approach for representing context and performing word sense disambiguation using neural networks. *Neural Computation*, 3(3):293–309.

[46] Gallant S. Context vector representations for document retrieval. In *AAAI Natural Language Text Retrieval Workshop*, 1991.

[47] Kaski S. Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *In Proceedings of the International Joint Confer- ence on Neural Networks, IJCNN'98*, pages 413–418. IEEE Service Center, 1999.

[48] Robertson S. and Spärck Jones K. Simple, proven approaches to text retrieval. 1997.

[49] Amit Singhal, Gerard Salton, Mandar Mitra, and Chris Buckley. Document length normalization. *Inf. Process. Manage.*, 32(5):619–633, 1996.

[50] Dill Stephen, Nadav Eiron, David Gibson, Daniel Gruhl, R. Guha, Anant Jhingran, Tapas Kanungo, Sridhar Rajagopalan, Andrew Tomkins, John A.Tomlin, and Jason Y. Zien. Semtag and seeker: Bootstrapping the semantic web via automated semantic annotation. In *Proceedings of the Twelfth International Conference on World Wide Web (WWW-2003), Budapest, Hungary*, pages 178–186, 2003.

[51] Peter Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. 2001.

[52] Peter D. Turney. Measuring semantic similarity by latent relational analysis, 2005.

[53] Peter D. Turney and Patrick Pantel. From frequency to meaning: Vector space models of semantics. *CoRR*, abs/1003.1141, 2010.

[54] Cornelis Joost van Rijsbergen. *Information Retrieval*. 1979.

[55] Weaver Warren. *Machine Translation of Languages*. New York: John Wiley & Sons, 1949.

[56] X. Wei, F. Peng, and B. Dumoulin. Analyzing web text association to disambiguate abbreviation in queries. In *In Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '08)*, New York, NY, 2008. ACM.

[57] Dominic Widdows. *Geometry and Meaning*. CSLI Publications, Stanford, California, USA, 2004.

[58] Harris Z. *In Papers in structural and transformational Linguistics*, chapter Distributional structure, pages 775–794.

List off Tables

List of Abbreviations

Appendices

9. List of filter words for Czech language

[dnes cz timto budes budem byli jses muj svym ta tomto tohle tuto tyto jej zda proc mate tato kam tohoto kdo kteri mi nam tom tomuto mit nic proto kterou byla toho protoze asi ho nasi napiste re coz tim takze svych jeji svymi jste aj tu tedy teto bylo kde ke prave ji nad nejsou ci pod tema mezi pres ty pak vam ani kdyz vsak ne jsem tento clanku clanky aby jsme pred pta jejich byl jeste az bez take pouze prvni vase ktera nas novy tipy pokud muze design strana jeho sve jine zpravy nove neni vas jen podle zde clanek uz email byt vice bude jiz nez ktery by tere co nebo ten tak ma pri od po jsou jak dalsi ale si ve to jako za zpet ze do pro je na]