Unsupervised Word Sense Disambiguation with Sense Embeddings

Unüberwachte Wortbedeutungsdisambiguierung mit Embeddings von Bedeutungen
Master-Thesis von Maria Pelevina
Tag der Einreichung:

1. Gutachten: Prof. Dr. Chris Biemann
2. Gutachten: Dr. Alexander Panchenko

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Unsupervised Word Sense Disambiguation with Sense Embeddings
Unüberwachte Wortbedeutungsdisambiguierung mit Embeddings von Bedeutungen

Vorgelegte Master-Thesis von Maria Pelevina

1. Gutachten: Prof. Dr. Chris Biemann
2. Gutachten: Dr. Alexander Panchenko

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den August 16, 2016

(Maria Pelevina)

Contents

1 Introduction

One of the considerable challenges in natural language understanding, the ability of computers to derive meaning from human language, is lexical ambiguity. There are words in human language that have the same spelling, but relate to different concepts. Consider the following example:

The whole family was sitting at the table$_1$ and enjoying their dinner.
This table$_2$ presents the preliminary results of our investigation.

The word *table*$_1$ refers to a piece of furniture, while the word *table*$_2$ means a type of data arrangement in rows and columns. Humans have an ability to identify the implied meaning of an ambiguous word from the context it is used in. For example, the words *sitting* and *dinner* indicate to the first (furniture) meaning of the word *table*$_1$. In order to provide computers with this ability, we have to algorithmically imitate the same process that happens in human's brain when they read sentences like the ones above. In the field of natural language processing (NLP) this ability is referred to as word sense disambiguation (WSD). In this thesis we focus on building a WSD system operating on word and sense embeddings.

1.1 Motivation

Word embeddings are low dimensional representations of natural language words as real-valued vectors. They capture important semantic and syntactic features of words in a compact manner and help to overcome the curse of dimensionality. High-quality word embeddings exhibit many interesting properties on their own and were shown to improve performance in several natural language processing (NLP) tasks, when used as input features. Since introduction of computationally efficient training algorithms in the last years, word embeddings and their use remain on peak of interest from the research community.

Word embeddings rely on the co-occurrences of the words with their context (surrounding words). This approach is based on the Distributional Hypothesis introduced and popularized in the 1950s by English linguist J. R. Firth. It states that words that are used and occur in the same context tend to have similar meaning. Indeed, two semantically similar words like *lift* and *elevator* do not share any common features if we consider them as a sequence of characters or use morphological information. However, they both appear next to *up*, *down*, *stairs* and *floor*. Exactly these regularities are captured by values of the vectors representing these two similar words.

Continuous vector space language models place similar words close to each other and pull unrelated words apart. This explains one of the most discussed property of word embeddings. A query for the nearest neighbours of *France* returns words like *Germany*, *Spain* and *Italy*. Moreover, embeddings capture analogy relations between words which can be revealed with algebraic operations:

$$vec(France) - vec(Paris) \sim vec(Germany) - vec(Berlin)$$

(semantic regularities) and

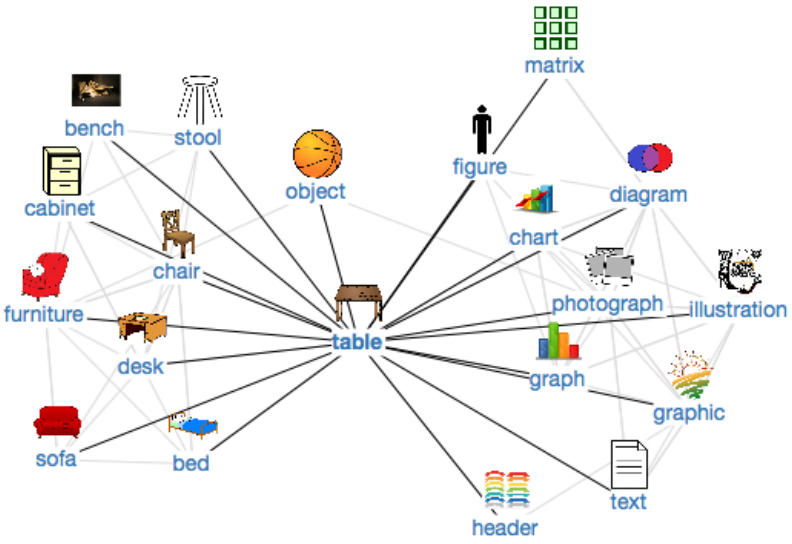


Figure 1.1: Word similarity graph of the word *table* based on vector cosine similarity.

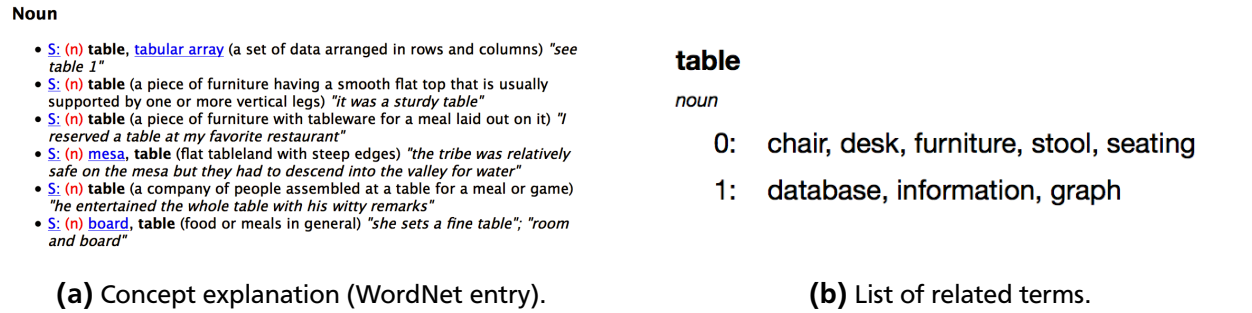
$$vec(car) - vec(cars) \sim vec(cat) - vec(cats)$$

(singular/plural regularities) [Mikolov et al., 2013d].

Recently several simple and highly computationally efficient models have been introduced that can train word embeddings with neural networks on complete English Wikipedia in several hours. Most notable among them are Continuous Bag Of Words (CBOW) and Skip-Gram models [Mikolov et al., 2013a], commonly known as a *word2vec* toolkit and GloVe [Pennington et al., 2014]. They have inspired a lot of research into the application of word embeddings for various NLP tasks. For example, they were successfully used to improve state-of-the-art results of the named-entity recognition (NER) task [Passos et al., 2014], dependency parsing [Bansal et al., 2014] and machine translation [Mikolov et al., 2013b].

A common deficiency of mentioned word embeddings model is that they do not account for the word *ambiguity* (polysemy or homonymy) and learn a unique representation per word. This representation mixes several, sometimes unrelated senses of an ambiguous word into one vector. If we retrieve 20 neighbours of the word *table* based on cosine vector similarity and build an word similarity graph of those neighbours, it will look similar to the one in Figure 1.1. While this is acceptable in some NLP tasks like part of speech (POS) tagging or NER, it will be misleading in several other scenarios, for example machine translation. In particular, the translation of the word *table* into German depends on its semantic: *der Tisch* for $table_1$ and *die Tabelle* for $table_2$. Other applications that could benefit from the correct representation of word senses include information retrieval and sentiment analysis.

It is desirable to represent each sense of a words as a separate vector. In scope of these thesis we are going to develop a method that produces multi-sense word embeddings (also referred to as multi-prototyped word embeddings or sense vectors) and we will show how to use those embeddings for disambiguation of polysemous words in context. A detailed definition of the task follows in the Section 1.2. An overview of related work will be given in Chapter 2.



(a) Concept explanation (WordNet entry). **(b)** List of related terms.

Figure 1.2: Sense representation types.

1.2 Problem Statement

The problem of learning multi-prototyped word embeddings can be regarded as a special case of word sense induction (WSI) problem. It is also closely related to word sense disambiguation (WSD) task. In this work we wish to address these problems together, as it will provide us a method to asses the quality of produced sense embeddings.

The main objective of this thesis can be summarized in two steps:

1. induce word senses and associate them with sense-specific embeddings,
2. use word and sense embeddings to disambiguate words in context.

Requirements on Word Sense Induction

WSI is the task of automated identification of word senses from plain corpus. Its product is a *sense inventory*: a mapping from words to the list of their senses. The senses may be formalised in different ways, for example as a concept explanation like in dictionaries or as a set of related terms (see Figure 1.2 for comparison). They may also differ in the way they were constructed: either manually by language experts or automatically, usually with some form of word or context clustering. Finally, sense inventories differ in terms of granularity and domain. Some of the most prominent hand-crafted sense inventories designed to be used in computational manner for all sort of NLP tasks are WordNet for the English language [Fellbaum, 1998], GermaNet for the German language [Hamp and Feldweg, 1997] and their multi-lingual counterpart BabelNet [Navigli and Ponzetto, 2012].

With regard to these properties of sense inventories we set the following requirements on the WSI system:

- each word sense is represented by a unique sense vector

  Sense inventory in form of vectors facilitates its use for NLP application and allows to profit from advantages of word embeddings. Whenever necessary, such inventory can be easily transformed into a human-readable format by listing nearest neighbours for each sense vector.

- senses are induced in a fully unsupervised and knowledge-free manner

  Constructing any kind of a lexical resource by hand is a time-consuming and expensive process. As a result, it makes any system relying on such resources de-facto restricted to 3-5

most popular and intensively researched languages. Moreover, hand-crafted sense inventories have difficulties to catch up with an ever-changing language, for example the widely used lexical database WordNet does not list senses like *tablet* as *portable computer* yet. Unsupervised and knowledge-free approach will ensure the applicability of our method to a wide variety of languages and specialised domains.

- coarse sense distinction

  It has been observed that both experts and non-experts have difficulties to reach high level agreement in disambiguation using fine-grained inventories like WordNet. The difference between senses is too subtle and may not actually reflect how non-experts understand word meaning. Instead we opt for a coarse-grained inventory as a more practical option [Brown, 2008]. Figure 1.2 shows difference between fine-grained (on the left) and coarse-grained (on the right) sense inventories.

- broad word coverage

  We want a model that allows to induce senses for the whole vocabulary of the corpus and not only for a limited number of specific words.

Requirements on Word Sense Disambiguation

Word sense disambiguation problem can be formulated as follows: given a word w and its context c return $s \in S$, where S is a set of senses of word w according to some sense inventory. We say that a WSD method is supervised, if it requires a model training step on data with sense-annotated words. We also call it knowledge-based if it disambiguates words with respect to and/or using clues from an external hand-crafted knowledge resource like dictionaries, WordNet or Wikipedia disambiguation pages. In contrast, a knowledge-free WSD performs disambiguation with respect to an automatically induced inventory. Our requirements for this step are:

- unsupervised disambiguation

  Even though sense-annotated data is available, it is very rare, often limited in word coverage and bound to a specific sense inventory. As we have argued earlier for induced sensed inventories, the construction of hand-annotated datasets for every new inventory would very expensive and practically infeasible process. Therefore our requirement is to develop an unsupervised WSD approach.

- knowledge-free disambiguation

  The disambiguation method should not be bound to any specific sense inventory. This will ensure the applicability of our model to various languages and domains, as long as an induced or hand-crafted inventory of suitable format is provided.

To wrap up this section we summarise the main objective of this work as follows:

> The goal of this thesis is to build and evaluate an unsupervised knowledge-free word sense induction and disambiguation (WSID) system using sense embeddings.

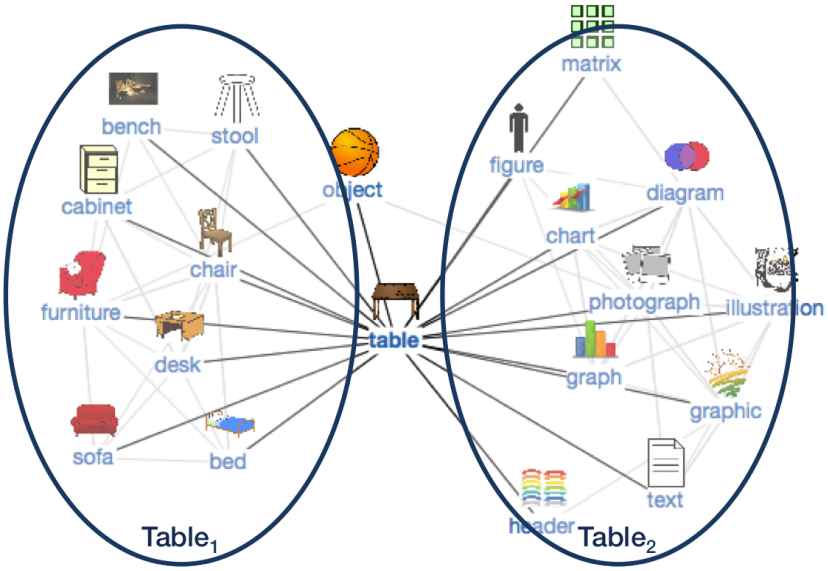


Figure 1.3: Two clusters representing senses of the word *table*.

1.3 Hypothesis

Word sense induction

Our approach for sense induction is based on clustering of local neighbourhoods of words. It is known that vector space models place semantically similar words close to each other and pull unrelated words apart. We have shown in Figure 1.1 that in a setting where there is only one vector for each spelled word (single-prototyped vector space model), a collection of nearest words for a polysemous word includes terms related to its different senses. We observe that neighbours corresponding to one sense of the word table (*desk*, *chair*, *furniture*) tend to be closely related to each other and only loosely connected to words representing a second sense (*chart*, *graph*, *matrix*). We therefore suggest to cluster ego-networks of all words and to interpret clusters as senses. An ego-network is a word similarity graph consisting of a single node (ego) together with the nodes it is connected to (alters) and all the edges among those alters. For illustration see Figure 1.3. We further suggest that a sense vector may be calculated as an average of word vectors belonging to a cluster, thus solving our first problem.

Word sense disambiguation

Assuming that vector space models position words in space with respect to their similarity we propose to disambiguate words in context by mapping them to the vector space and comparing the similarity between context and sense vectors of a target word. A sense vector closest to the context is the answer. We expect that context words {*family*, *dinner*, *sitting*} are sufficient to identify a correct sense of *table* as furniture.

1.4 Contributions

The main contributions of this work are:

- An unsupervised and knowledge-free Word Sense Induction system based on clustering of word ego-networks that represents senses as sense embeddings.
- An integrated Word Sense Disambiguation mechanism that performs comparatively to the state-of-the-art systems.
- Systematic evaluation of proposed WSID system on publicly available datasets.

In addition to this, we provide the implementation of our systems, as well as several pre-trained models online.[1]

[1] https://github.com/tudarmstadt-lt/sensegram

2 Related Work

The goal of this thesis, as defined in the previous chapter, is to build a word sense induction and disambiguation system operating on sense embeddings. This challenge relates to two strains of research: multi-prototyped word embeddings and word sense disambiguation. In this Chapter we discuss them both.

2.1 Sense Embeddings

Single-prototype vectors are not able to distinguish different senses of ambiguous words. Since introduction of efficient learning algorithms for word embeddings, researches have intensively studied methods to overcome this problem. The distributional hypothesis provided theoretical foundation for classical word embeddings and similarly the key to learning sense-aware word representations lies in the word's contexts. This is in fact an obvious observation if we consider how humans can recognise a sense of a word only with regard to the context and not in isolation. So for example, we know that *bank* in a sentence *"A bank can only issue a card if you have sufficient funds on deposit.*" refers to a financial institution because we see it accompanied by words *funds* and *deposit*.

The difference of methods that learn sense embeddings lies in the way they incorporate contextual information: before or after training of embeddings, using clustering of contexts or probabilistic approaches to differentiate senses, relying on lexical resources or external knowledge bases or not. In following we give an overview of approaches that represent word senses as vectors, omitting any work that might perform sense induction, but does not bind it to vector space language models.

2.1.1 Context Clustering

Following the pioneering work of [Schütze, 1998], both [Reisinger and Mooney, 2010] and [Huang et al., 2012] explicitly pre-cluster contexts of each word to induce senses. Afterwards they generate a distinct prototype for each sense cluster. A sense prototype in [Reisinger and Mooney, 2010] is represented with a centroid of a context cluster. This approach does not operate on neural network learned word embeddings, but works with a tf-idf [Rajaraman and Ullman, 2011, p. 8] weighted counting model. Its clustering algorithm produces a fixed number of k clusters per word that according to the authors do not directly correspond to word senses. Instead they capture meaningful variations of word usage and allow to measure similarity of two words, for example, as an average similarity between all their cluster pairs. It has been evaluated on WordSim353 dataset [Finkelstein et al., 2002] for word similarity task and on a custom dataset for near-synonym prediction. Basically, this approach does not explicitly model word meanings, but rather decomposes a single word embedding into several ones to overcome a bias towards the most frequent sense.

[Huang et al., 2012] mostly follows the previous idea, but instead of tf-idf vectors it uses word embeddings trained with a custom neural language model that learns word representations from

local and global context. Another difference is that to produce sense vectors it relabels all word occurrences in the corpus with respect to the context cluster they belong to and retrains a neural network on this relabelled corpus. Similarly to [Reisinger and Mooney, 2010] the number of senses is fixed and in this case also does not directly represents word senses. The quality of embeddings was evaluated on the WordSim353 dataset and on a custom Stanford's Contextual Word Similarity (SCWS) set, where word pairs are enriched with contextual information.

2.1.2 Extensions of Skip-Gram

Systems described in this subsection share a common property: they jointly perform word vector learning and sense discrimination, directly producing word and sense embeddings in one step. Also they all build on the Skip-Gram model of [Mikolov et al., 2013a].

[Tian et al., 2014] models word ambiguity from a probabilistic perspective. In comparison to the Skip-Gram model, each word is modelled as a set of vectors and probability of two words' co-occurrence is conditioned on the appearance of words' vector. Each word has a fixed number of prototypes, however in this case each prototype is associated with its prior probability. Most probable prototypes correlate well with word senses. This model has been evaluated on the word similarity in context task using the SCWS dataset. It performs comparably with [Huang et al., 2012], but runs about 3 times faster. It outperforms Skip-Gram vectors on the same task, but most probably requires longer training time due to a more complex model.

[Neelakantan et al., 2014] were the first to present a non-parametric Multi-sense Skip-Gram (NP-MSSG) model that discovers a varying number of senses per word type. During online training they compare current context with already available sense vectors, and either update the closest sense vector or create a new one. A sense slit-off happens if the distance from the context to the nearest sense is lower than some threshold. They evaluate both parametric (in which new vectors are never created) and non-parametric versions of their model on word similarity task, showing that their approach beats both original Skip-Gram model and [Huang et al., 2012]. Even though the non-parametric version is able to induce a reasonable number of senses, the parametric version with 3 fixed senses per word performs better or comparably in the majority of tests. Quite importantly, this model is highly computationally efficient: it does not run significantly slower in comparison to the original Skip-Gram (2 hours vs. 0.33 hours) and improves impressively over [Huang et al., 2012] (2 hours vs. 168 hours).

[Li and Jurafsky, 2015] propose a solution that incorporates Chinese Restaurant Processes (CRP) into the Skip-Gram model and investigate the real value of multi-sense embeddings in NLP applications in comparison to single-prototyped word vectors. The underlying idea behind their approach is very similar to that of [Neelakantan et al., 2014]: allocate a new representation for a word if existing ones cannot explain it well enough. The main difference is that this model splits off a new sense vector with a chance counter-proportional to probability of current context co-occuring with already known senses and their frequency, while NP-MSSG makes decision based on a threshold. Evaluation on word similarity (SCWS dataset) has shown that this variation achieves better results than [Neelakantan et al., 2014] and [Chen et al., 2014], although the advantage is minimal. Furthermore, they evaluate sense vectors on POS-tagging, named entity recognition and several sentiment analysis tasks, using sense vectors as input features. They observe that multi-sense embeddings do indeed offer superior performance on some tasks (POS-tagging, semantic relatedness), but not on others (NER, sentiment analysis). They come to a conclusion, that importance

of sense disambiguation for some applications might be overrated and that in some cases, similar improvement can be achieved simply with increase of vector dimensionality.

To the best of our knowledge, the highest performance results at this moment are achieved by the AdaGram model presented in [Bartunov et al., 2015]. They are also the first to apply sense vectors to word sense disambiguation task. Their model is a bayesian extension of Skip-Gram with Dirichlet process (DP) used for automatic determination of the required number of prototypes. The sense vectors represent different word senses and granularity of sense induction is configured by a hyperparameter. AdaGram is compared to [Neelakantan et al., 2014] and [Tian et al., 2014] on several WSI and WSD test datasets (SemEval-2007 Task 2, SemEval-2010 Task 14, SemEval-2013 Task 13 and a custom Wikipedia-based set). First of all, AdaGram consistently and significantly outperforms other models on all sets. Secondly, it is observed that a parametric version of [Neelakantan et al., 2014] (fixed three senses per word) shows better results than a non-parametric one, which leads authors to a conclusion, that their systematic Bayesian approach is better suited for word sense induction. Unfortunately, there is no comparison to [Li and Jurafsky, 2015]. It is worth noticing, that in word similarity AdaGram performs worse than high-dimensional vectors of Skip-Gram and both NP-MSSG and MSSG, however it is argued, that this task is not perfectly suitable for assessment of quality of sense embeddings.

2.1.3 Knowledge-based and Miscellaneous

Finally, there is a group of methods that either rely on external knowledge bases, or find another unique way to learn sense distinct word embeddings. [Qiu et al., 2014] propose to learn a separate vector for every part of speech of a term. While this might be a natural approach on its own, it only partially solves the WSD problem, as different word senses do not necessarily correspond to different part of speech roles of a word in context. Moreover, as part of their method they first process a corpus with a POS-tagger and then train word representations for POS-tagged terms. So essentially the POS disambiguation strength of their system is limited by the quality of a POS-tagger.

[Chen et al., 2014] learn sense embeddings for WordNet [Fellbaum, 1998] *lexemes*. To do so they average vector representations of most relevant words in a respective sense glosse (sense explanation). Clearly, the number of senses is defined by the WordNet inventory, even though it is quite easy to add a new sense to the model once it appears. The WSD step with this model is performed as a search for maximum cosine similarity between averaged context vectors and known sense vectors of the target word. The method is compared to winners of SemEval-2007 competitions with mixed results, showing no decisive superiority of their approach.

The AutoExtend method [Rothe and Schütze, 2015] addresses a similar objective from another perspective. Inspired by the mathematical properties of the embeddings discussed by [Mikolov et al., 2013d], they suggest that in a vector space a word is a sum of its *lexemes* in WordNet, and analogously, a *synset* (set of interchangeable synonyms) is a sum of its lexemes. Based on this assumption they propose a neural network architecture that learns synsets and lexemes embeddings by taking a graph of WordNet entities and any available word embeddings as input. This method does not require an access to the corpus on which the word embeddings where trained and can be transferred to learn entities embeddings of other knowledge bases, such as Freebase. In this scenario word sense representations correspond to lexemes embeddings. The method has been evaluated on word similarity (SCWS dataset) and word sense disambiguation (Senseval-2 and Senseval-3) tasks, showing improvement over the state-of-the-art results.

[Liu et al., 2015] train topic specific embeddings by combining Skip-Gram with latent Dirichlet allocation (LDA) topic models. Combination of jointly learned word vectors and topic vectors can be interpreted as senses. They improve performance on text classification task as compared to single-prototyped word embeddings. [Wu and Giles, 2015] refer to Wikipedia for sense information. Working on a "one sense per article" assumption, they cluster Wikipedia articles to induce word senses and then derive a separate sparse word vector for each word sense associated with a cluster. Finally, [Guo et al., 2014] use bilingual resources and induce senses by clustering of word's translations. Sense vectors are trained on a corpus that is sense-labelled with respect to induced clusters. To train vectors they use an earlier recurrent neural network language model of [Mikolov et al., 2010].

2.2 Word Sense Disambiguation

The goal of this section is to provide a brief overview over the types of word sense disambiguation systems and give examples of most prominent techniques. For a detailed and comprehensive survey refer to [Agirre and Edmonds, 2007] and [Navigli, 2009].

Navigli defines WSD as the *"ability to identify the meaning of words in context in a computational manner*". A meaning of a word is determined with respects to a particular sense inventory which contains information about word-sense distinction. While the goal of all such method is the same, they are distinguished according to the amount of external knowledge used and according to the level of supervision. Generally, they are categorised in three groups: knowledge-based, supervised and unsupervised methods.

Knowledge-based approaches

Such approaches utilize information from a hand-crafted, often human-readable lexical resources like dictionaries, WordNet or, since recently, Wikipedia. These resources also act as a sense inventory. The classical Lesk algorithm [Lesk, 1986] looks for overlap between the definition of a target word and definitions of context words. [Vasilescu et al., 2004] investigated variants of Lesk algorithm on Senseval-2 task and found out that (1) filtering of stop words from context improves performance and (2) context words further away from the target provide less useful clues for disambiguation. To overcome a lexical gap (common problem of overlap-based approaches, when the description and context do not have much vocabulary in common) [Miller et al.,] expand words with similar terms using distributional similarity. Another difficulty of knowledge-rich approaches is that lexical resources as WordNet are readily available only for popular languages as English. To address this issue the system *Babelfly* [Moro et al., 2014] performs entity linking (EL) and WSD using semi-automatically constructed lexical semantic network BabelNet [Navigli and Ponzetto, 2012] that covers 271 languages.

Supervised approaches

Supervised approaches use an explicitly sense-labeled training corpus to construct a model, usually building one model per target word. Decisions tree are one of the simplest approaches, in which context features like "bank *of?*" are used to choose a correct sense. However, other supervised WSD algorithms consistently outperform decision trees [Navigli, 2009, p. 17]. Several experiments tested "non-deep" neural networks for WSD, e.g. [Leacock et al., 1993, Mooney, 1996, Towell and Voorhees, 1998], but due to the lack of contemporary studies of these approaches we are unable to make any assumption on their performance. Later SVMs have been recog-

nised as the most successful mechanism for supervised WSD [Lee and Ng, 2002] to this date. While Naïve Bayes classifiers are outperformed by more powerful methods such as SVMs, they generally compare well despite their independence assumption [Mooney, 1996, Ng, 1997a, Leacock et al., 1998, Pedersen, 2007]. Instance-based models, also known as memory-based models, have also been found to be among the highest-performing methods on this task [Ng, 1997b, Daelemans et al., 1999]. Furthermore, ensembles of several supervised WSD systems surpass performance of each of the individual classifiers [Klein et al., 2002, Florian et al., 2002].

Unsupervised approaches

While knowledge-based and supervised approaches demonstrate top performance in competitions, they suffer from the knowledge acquisition bottleneck. Supervised approaches require considerable amount of sense-annotated training data which is expensive to create and often inconsistent. Knowledge-based approaches a tied to the fixed number of senses in the inventory. This is problematic as (1) senses emerge and disappear over time; (2) different applications require different granulates of a sense inventory.

Unsupervised WSD approaches rely neither on hand-annotated sense-labeled corpora, nor on handcrafted lexical resources. Firstly, they automatically induce a sense inventory from raw corpora. Such unsupervised sense induction methods fall into two categories: context clustering, such as [Pedersen and Bruce, 1997, Schütze, 1998, Reisinger and Mooney, 2010, Neelakantan et al., 2014, Bartunov et al., 2015] and word similarity graph (ego-network) clustering, such as [Lin, 1998, Pantel and Lin, 2002, Widdows and Dorow, 2002, Biemann,]. Secondly, they use disambiguation clues from the induced sense inventory for word sense disambiguation. Usually, the WSD procedure is uniquely defined by the design of sense inventory. It might be the highest overlap between the instance's context words and the words of the sense cluster, as in [Hope and Keller, 2013a] or the smallest distance between context words and sense hubs in graph sense representation, as in [Véronis, 2004]. If senses are represented by distinct embeddings, the maximum cosine similarity between a sense vector and context vector representation can be used to choose a correct sense.

Our method belongs to the group of unsupervised approaches using ego-network clustering for sense induction.

3 Background

In the introductory chapter to this thesis we have put forward a hypothesis that single-prototype word embeddings can be transformed into sense-specific vectors via clustering of word similarity graphs. The approach we propose to complete this task requires sequential execution of following steps: (1) computation of word embeddings, (2) construction of word similarity graphs, and (3) clustering of such graphs. All these tasks are subjects of long-standing research with multiple solutions proposed over the years. We are going to select suitable systems for each step among existing ones and combine them to build our custom WSID system. In order to justify our choices and to provide the reader with the background knowledge necessary for understanding of further chapters, we are going to introduce existing approaches that are most relevant to our work.

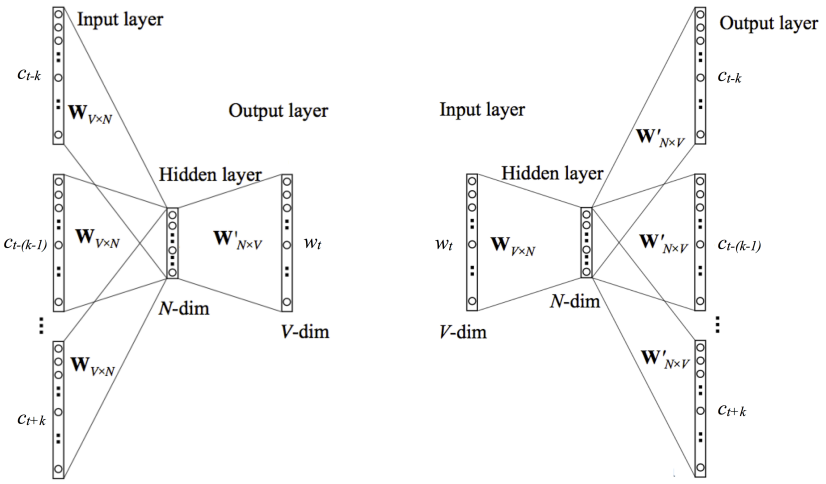
In the first section we talk about word embeddings and the most efficient algorithms for their training. In Section 3.2 we discuss proposed measures for semantic word similarity and systems for its computation. In the last section we present clustering algorithms used for sense induction.

3.1 Word Embeddings

Word embeddings are distributional representation of words with low-dimensional vectors, typically of 100 – 500 dimensions. They are opposed to sparse high-dimensional vectors of length 10,000 – 100,000. Independently of the length, word vector representations are based on a distributional hypothesis developed in the works of Zellig Harris, John Firth and Ludwig Wittgenstein during the 1950s [Firth, 1957]. The hypothesis states that we can determine the meaning of a word by looking at the context it appears in. The objective is that the relative similarity of these vectors correlates with semantic similarity of corresponding words.

Variety of models for distributed vector representation differ in types of contextual information they use and ways statistical information of corpus is modelled. In the traditional scenario, local word contexts are used to compute a sparse square matrix of word co-occurrences, where entries represent the number of times two words appeared together in a fixed-sized word window. Word association measures such as Pointwise Mutual Information (PMI) [Church and Hanks, 1990] are often used instead of raw counts. The Latent Semantic Analysis (LSA) [Deerwester et al., 1990] developed in information retrieval, uses documents as contexts. In both of these approaches, dimensionality reduction techniques such as Singular Value Decomposition (SVD) can to be used to reduce the length of sparse word representations. Generally such models are called *count-based*.

Another way of learning word vectors is to train a neural network that tries to predict a next word given its predecessors. As a by-product of this task we get weight matrices of the trained network, in which rows/columns correspond to words and turn out to represent their semantics very well. The advantage of this approach is that it helps to avoid the dimensionality reduction step and directly produces low-dimensional word vectors. First models were proposed by [Bengio et al., 2003, Morin and Bengio, 2005, Collobert and Weston, 2008], but they suffered from considerable computational complexity caused by sophisticated non-linear architectures of networks. Mikolov et al. [Mikolov et al., 2013a] made it practical to learn word embeddings by



(a) CBOW model **(b)** Skip-Gram model

Figure 3.1: Architectures of *word2vec* models.[1]

suggesting a simplified architecture of a neural network with only three layers. The models of this group are referred to as *predictive neural network models*.

There is no fundamental difference between predictive neural network and count-based language models, which are rather different computational means to arrive at the same type of a semantic model [Levy and Goldberg,]. In the rest of this section we are going to discuss two state-of-the-art models, each representing one of these categories.

3.1.1 word2vec

word2vec is a toolkit implementing two predictive neural network models for training of word embeddings.

Theory

Mikolov et al. decided to explore simpler architectures of neural networks that could be trained much faster on more data, without loosing in quality of the vectors. Two new architectures were proposed: a *Continuous Bag-of-Words* (CBOW) model and a *Skip-Gram* model. Both operate on $(w_t, \mathbf{c_t})$ training pairs consisting of a word w_t and the set of it's context words $\mathbf{c_t} = \{c_{t-k}, \ldots, c_{t-1}, c_{t+1}, \ldots, c_{t+k}\}$, where $2k$ is the size of the context window.

As depicted in Figure 3.1, the main difference between CBOW and Skip-Gram models is that:

- CBOW: predicts the word given its context,
- Skip-Gram: predicts the context given a word.

The neural network consists of three layers: input, hidden and output layers. Let V and N be the size of a corpus vocabulary and the number of nodes in the hidden layer. The input words are encoded as one-hot vectors of length V and each row of weight matrix $\mathbf{W}_{V\times N}$ is an N-dimensional

[1] The image is adapted from [Rong, 2014]

vector representation $\mathbf{v}_{w_I}$ of an associated word w_I of the input layer. This implies that the activation function of the hidden layer is simply linear [Rong, 2014], which accounts for computational efficiency of the models. Between the hidden layer and the output layer there is another weight matrix $\mathbf{W}'_{N\times V}$, where each column $\mathbf{v}'_{w_O}$ corresponds to an output word w_O. At the output layer the models produce posterior multinomial distributions (one for CBOW and $2k$ for Skip-Gram), which are obtained using softmax function and defined as:

$$p(w_O \mid w_I) = \frac{e^{\mathbf{v}'_{w_O}\cdot\mathbf{v}_{w_I}}}{\sum_{j=1}^{V} e^{\mathbf{v}'_{w_j}\cdot\mathbf{v}_{w_I}}}. \tag{3.1}$$

At the end of the training the rows of $\mathbf{W}_{V\times N}$ matrix are used as word vectors representations.

Since the computation of the denominator in Equation 3.1 is very expensive due to the sum over all words in the vocabulary, authors suggested a *negative-sampling* approach. Instead of iterating over all possible word pairs for w_I, they propose to randomly sample some negative (not co-occurring with w_I) examples of words from the vocabulary and only use them.

word2vec models employ further techniques to discard some of the training instances [Goldberg and Levy, 2014]. First, words appearing less than a minimal count of times are not considered as either words or contexts. Second, very frequent words are down-sampled with the probability:

$$P(w) = 1 - \sqrt{\frac{t}{f(w)}},$$

where $f(w)$ is the frequency of a word w and t is a chosen threshold, typically around 10^{-5}. These words are removed from the text before training pairs are generated.

The combination of these optimization techniques results in extremely efficient training: an optimized single-machine implementation can train on more than 100 billion words in less than one day, while previously proposed models [Collobert and Weston, 2008, Turian et al., 2010] required several weeks on a smaller corpus [Mikolov et al., 2013c]. The quality of vectors is also said to be improved [Baroni et al., 2014]. According to the authors, Skip-Gram model trains better word representation when available corpus is small, while CBOW is faster and more suitable for large datasets [Mikolov et al., 2013b].

Properties

Word embeddings trained with *word2vec* exhibit several interesting properties. First of all, as expected from the training objective and analogous to the vectors obtained by other methods, the angle between vectors of two words corresponds to the human's perception of word similarity. To give an example, the cosine similarity on a scale $[-1,+1]$ for words *student* and *university* may be 0.7, whereas for *student* and *ship* it may be 0.2.

Furthermore, *word2vec* vectors have a property of *additive compositionality* [Mikolov et al., 2013c]. They exhibit linear structure that makes it possible to meaningfully combine words by element-wise addition of their vector representations. Several examples of nearest neighbours of two-word compositions are given below:

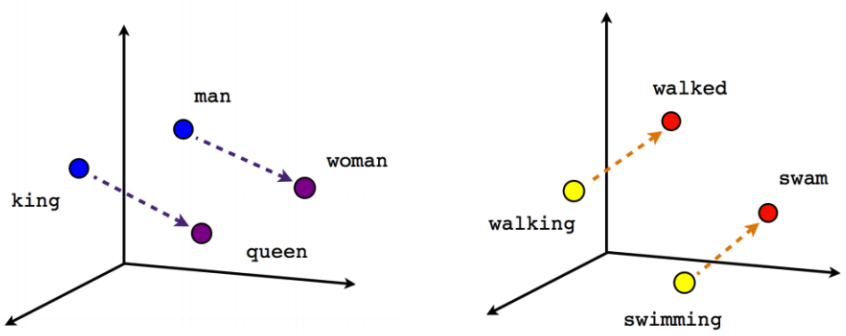


Figure 3.2: Visualisation of gender relationship (on the left) and verb tense relationship (on the right) between vector pairs using t-NSE dimensionality reduction technique.[2]

$$vec(house) + vec(king) \rightarrow vec(palace), vec(castle), vec(prince)$$
$$vec(Germany) + vec(river) \rightarrow vec(Elbe), vec(Rhine), vec(Danube)$$
$$vec(grapes) + vec(alcohol) \rightarrow vec(wine), vec(wines), vec(brandy)$$

Finally, the vectors are able to implicitly learn semantic and syntactical relationships between words. These regularities are observed as constant vector offsets between pairs of words sharing a particular relationship. For example, as illustrated in the Figure 3.2:

$$vec(man) - vec(woman) \sim vec(king) - vec(queen),$$
$$vec(swimming) - vec(swam) \sim vec(walking) - vec(walked),$$

vectors can capture gender and verb tense relationships between word pairs. Other observed relationships include singular/plural for nouns, comparative/superlative for adjectives, as well as country/capital and country/currency semantic relations. This allows to solve word analogy tasks such as *car:cars cat:X* by computing $X = vec(cars) - vec(car) + vec(cat)$ and taking the nearest vector of X. However, according to the experiments of Prof. Jordan S. Ellenberg presented in his blog post *Messing around with word2vec*[3], the vectors are not able to find antonyms of words in a task like: *waste:economise happy:X*.

The important takeaway from this paragraph is that (1) cosine similarity between word embeddings reflects, at least partially, semantic word similarity and (2) linear combinations of vectors capture relationships between concepts. While these examples were produced with vectors trained by *word2vec* toolkit, it is important to mention that the same properties were demonstrated by vectors trained using other architectures [Levy and Goldberg, 2014b, p. 2].

[2] The image is taken from *Vector Representations of Words* tutorial published on `https://www.tensorflow.org/versions/r0.7/tutorials/word2vec/index.html`

[3] `https://quomodocumque.wordpress.com/2016/01/15/messing-around-with-word2vec/`

Implementation

There exist two implementations of training algorithms for CBOW and Skip-Gram models. The first one is the original *word2vec* toolkit[4] written in C, which is fast and well parallelizable. Another one is a re-implementation of the same algorithms in Python[5] by Radim Řehůřek, distributed within the *gensim* package [Řehůřek and Sojka, 2010]. It provides a better interface for usage of trained vectors, but is several times slower. Therefore, it is advisable to train vectors with C implementation, but it is easier to "play" with them when loaded with *gensim* package.

The most relevant parameters for the training of word vectors are:

- `size`: dimensionality of word vectors. Influences the quality of representations. Common numbers are between 100-600.
- `window`: half-size of the symmetrical context window. Usually between 3 and 8.
- `sample`: threshold t for down-sampling of very frequent words. Reasonable threshold equals 10^{-5}.
- `negative`: number of negative examples used for each input word in Equation 3.1. Lower values lead to faster computation, but affect vector quality. Common values are between 3 and 10.
- `min-count`: threshold for removal of infrequent words from the corpus. Decreases model size and removes noisy/erroneous words.

Parameter names correspond to the *word2vec* interface of the training algorithm.

3.1.2 GloVe

GloVe is a word embeddings model presented in [Pennington et al., 2014], which uses explicit matrix factorisation to learn vectors. It belongs to the group of *count-based* methods. In the following we briefly explain the approach and how its performance compares to *word2vec* models discussed earlier.

The GloVe model learns vectors by examining the co-occurrences of words in a text corpus. These are encoded as a $X_{V\times V}$ matrix, where V is the size of corpus vocabulary and an entry X_{ij} is the number of times word i occurs in context of word j. The goal is to learn two separate sets of word vectors $\mathbf{w} \in \mathbb{R}^d$ and context word vectors $\tilde{\mathbf{w}} \in \mathbb{R}^d$, such that:

$$\mathbf{w_i} \cdot \tilde{\mathbf{w}}_\mathbf{j} + b_i + b_j = log(X_{ij}),$$

where b_i and b_j are scalar bias terms associated with words i and j and trained together with word/context vectors. The model is fit to minimize the sum of all squared errors using stochastic gradient descend algorithm. To balance the influence of very frequent and very rare word pairs, the errors are weighted with the function:

$$f\left(X_{ij}\right) = \begin{cases} \left(\frac{X_{ij}}{x_{\max}}\right)^{\alpha} & \text{if } X_{ij} < x_{\max} \\ 1 & \text{otherwise.} \end{cases}$$

[4] `https://code.google.com/p/word2vec/`
[5] `https://radimrehurek.com/gensim/models/word2vec.html`

For example, with $x_{\max} = 100$ it will treat equally any pair co-occurring more than 100 times and decrease the relevance of pairs observed together less than 100 times proportionally to their X_{ij}.

The authors suggest to use a mean of $\mathbf{w_i}$ and $\tilde{\mathbf{w}}_\mathbf{i}$ as a vector for a word w_i. Vectors trained with this model exhibit all the same properties that were discussed for *word2vec* models earlier in this section. Therefore, they can also be used to solve word similarity and word analogy tasks. In the original paper authors have reported a significant performance improvement of GloVe vectors over Skip-Gram and CBOW vectors on these tasks. However, Levy et al. conducted experiments comparing *count-based* and *predictive* word vector models (including *word2vec* and GloVe), which showed that performance differences between the methods are mostly local and insignificant, and that "*there is no consistent advantage to one algorithmic approach over another*" [Levy et al., 2015]. With regard to the training time the models are also comparable, however *word2vec* requires less memory.

3.2 Word Similarity

As we have mentioned in the beginning of this chapter, we want to learn sense-specific vectors via clustering of word similarity graphs. To build such graphs we need a measure for semantic word similarity. One measure that we could use is provided by vector space models discussed in the previous section. There the similarity between two words is defined as a cosine similarity of their vector representations. An alternative to this has been proposed by Biemann et al. [Biemann et al., 2013]. In their *JoBimText* framework each word is represented with a set of its observed context features and the similarity between two words is defined by the overlap of their feature sets.

Both approaches take their roots in the *Distributional hypothesis* as they find word characteristics in the context. In this section we present the word similarity measures in terms of both approaches, with an emphasis on the *JoBimText* modelling of distributional semantics, which has not been explained so far.

3.2.1 Similarities using Vector Space Models

In vector space models, including previously discussed *word2vec* and *GloVe* models, the similarity between words is defined as a cosine similarity of their vectors:

$$sim_{w2v}(w_i, w_j) = \frac{\mathbf{w}_i \cdot \mathbf{w}_j}{||\mathbf{w}_i|| \cdot ||\mathbf{w}_j||}, \tag{3.2}$$

where $\mathbf{w}_i$ and $\mathbf{w}_j$ are word embeddings of words w_i and w_j respectively. The range of the measure is $[-1, +1]$, with most of the word similarities distributed between -0.2 and 0.8.

3.2.2 Similarities using JoBimText

JoBimText is a framework for application of Distributional Semantics using lexicalized features. It provides a software solution for automatic text expansion and word sense induction using contextualized distributional similarity. Context features are used for the computation of similarities. It is highly scalable and open-source.

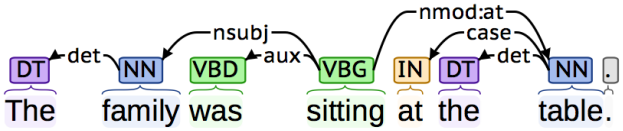


Figure 3.3: Dependency parse of example sentence from Stanford CoreNLP.[6]

Jo	Bim
table	case(•, at)
table	det(•, the)
table	nmod:at(sitting, •)
family	det(•, the)
family	nsubj(sitting, •)

Jo	Bim
table	trigram(the, •, was)
family	trigram(the, •, .)

Table 3.1: Dependency features (on the left) and trigram features (on the right) of words *table* and *family* extracted from the sentence in Figure 3.3.

A context feature of a word is some observation of this word in its context. In particular, *JoBimText* uses dependency and n-gram features. Dependencies are grammatical relations among words in a sentence and n-grams are word sequences of length n. To illustrate this consider an exemplary sentence in the Figure 3.3. Context features that can be extracted from this sentence for words *family* and *table* are listed in Table 3.1. The head word for which a feature is extracted is called *Jo* and the features are called *Bims*, hence the name of the framework. An arbitrary symbol '•' acts as a placeholder for the head word and enables features to be shared by different words. Thus, the semantic relatedness between words *desk* and *table* can be expressed by their sharing of a feature *nmod:at(sitting, •)*.

Features like these are extracted from a large corpus and aggregated by head words. The count statistics (number of times each *Jo*, each *Bim* and each *JoBim* pair appear in the corpus) are collected as well. To reduce the noise and to make the computation scalable, each feature of a word is ranked according to its significance for this word. Only p most significant features are used to represent a word.

The similarity between two words is then defined as the number of features they share:

$$sim_{JBT}(w_i, w_j) = features(w_i) \cap features(w_j). \tag{3.3}$$

This measure ranges from 0 to p.

The most influential parameters for this model are:

- features type: dependencies (Stanford parser [Klein and Manning, 2003], Malt parser [Nivre et al., 2006]) or n-grams (bigrams, trigrams).
- significance measure for *JoBim* pairs: Pointwise Mutual Information (PMI), Lexicographer's Mutual Information (LMI) or Log-likelihood ratio (LL). Definitions are given in [Biemann et al., 2013, p. 74].
- p: number of features per word. Optimal values lie between 1,000 and 5,000 for very large corpora.

[6] The image is taken from web visualizer at `http://nlp.stanford.edu:8080/corenlp/process`.

3.3 Word Clustering

This section is about clustering of graphs. Now that we have found several measures to express semantic similarity between words, we can use them to build local word similarity graphs. Such graphs are constructed for each word separately. They consist of (1) words most similar to the target word as nodes and (2) similarity relations between word pairs as weighted edges. The similarity measure we can use (sim_{w2v} and sim_{jbt} defined in equations 3.2 and 3.3) are symmetric, therefore we work with undirected graphs. Out hypothesis is that:

1. local similarity neighbourhood of a polysemous word includes words related to its different senses,

2. neighbours related to the same sense are stronger interconnected with each other than with words related to other senses.

If this hypothesis is true, then clustering of a word similarity graph should produce clusters corresponding the target word's senses.

In this section we explore two algorithms for graph clustering. Both of them do not need a predefined number of clusters to be found, which is a prerequisite for a reasonable sense induction system.

3.3.1 Markov Cluster Algorithm

The Markov cluster algorithm (MCL) is a graph clustering algorithm developed by Stijn van Dongen during his PhD [van Dongen, 2000]. It is based on a *random walker* concept, which is a theoretical foundation for several other clustering algorithms as well. If we consider a graph that can be clustered, it is supposed to have many links within a cluster and fewer links between clusters. Therefore, if we start a random walk from any node, we are more likely to stay within a cluster, than travel between.

MCL algorithm starts with a calculation of a square transition probability matrix, which models the random walker's chance of getting from one node to another in one step. Then it alternates *expansion* and *inflation* steps until convergence to discover where the flow tends to gather, and therefore, where clusters are.

The *expansion* step simulates the advancement of random walks. For example, taking the 2nd power of initial transition matrix results in a matrix with probabilities of a random walk from one node to another in two steps. This step allows to connect different graph regions.

The *inflation* step emphasises the flow discovered in the expansion step: strong connections are further strengthened and weak connections are demoted. This is performed by raising each matrix entry to a positive power r and subsequently re-normalizing columns. The power r is called an *inflation parameter* and it controls the extend of strengthening and weakening of the links. Eventually it influences over the granularity of clusters: higher values of r lead to finer-grained clustering.

During alternation of these steps the connections weakened below some threshold are deleted. Eventually the graph is separated into different segments, which are interpreted as clusters. The global convergence of the algorithm is not proven, however, experimental results show that the matrix starts to converge noticeably after 3-10 iterations in most of the cases. In the resulting

clustering each node is assigned to one cluster only, Exceptions to this are possible, but they occur only in parts of the graph which are symmetrical.

The worst case time complexity of the algorithm is $O(N^3)$, where N is the number of nodes. In practice it comes down to $O(N \cdot k^2)$, where k is the maximum number of non-zero elements (remaining connections) per column. Because the matrix becomes sparse after several first iterations, MCL is considered fast.

3.3.2 Chinese Whispers

Chinese Whispers (CW) is a randomized clustering algorithm proposed in [Biemann,]. It follows a bottom-up procedure: the algorithm starts with each node belonging to its own cluster and then iteratively increases the size of these clusters. Its time complexity is linear in the number of edges, which makes it very efficient, especially for sparse graphs. It can be used on undirected weighted graphs and produces a relatively coarse-grained hard partitioning (each node belongs to one class only).

The pseudocode of CW is given in Algorithm 1. It takes a graph as input and produces a class assignment of nodes. First, each node is assigned to a distinct class. Then a series of iterations is performed until the clustering stabilizes. In one iteration nodes are processed in random order and each node is assigned to the highest ranked class among its direct neighbours. Rank of a class c with respect to a node v is the sum of edges' weights connecting the node v with its neighbours of class c:

$$Rank(c, v) = \sum_{\substack{(v,u)\in E \\ class(u)=c}} weight(v, u). \tag{3.4}$$

Algorithm 1: Chinese Whispers algorithm.

input : $G = (V, E)$ – an undirected weighted graph
output: $class(v)$ – a cluster assignment for all nodes in V

foreach $v_i \in V$ **do**
  $class(v_i) = i$
end
while *clustering changes* **do**
  foreach $v \in V$ **do**
    $class(v) =$ highest ranked class among neighbours of v
  end
end

Intuitively this means that each node is assigned to the strongest class in the neighbourhood. Some alternative definitions of this function exist, which aim to decrease the influence of hub nodes or to set a minimum rank value necessary for an update to take place.

Because nodes are processed randomly, the algorithm is non-deterministic: each new run may produce a different partition. The time complexity of the algorithm is $O(n \cdot |E|)$. In worst case scenario, when the graph is fully connected, it is equivalent to the complexity of MCL (because $|E| = n^2$). In fact, CW can be regarded as a special case of MCL with very hard pruning of links: only one positive entry per row in transition matrix is kept.

It does not have an explicit parameter to control the granularity of clustering. However, it can be influenced by the density of the graph: with fewer edges in a graph an algorithm tends to create more clusters.

4 Word Sense Induction

In the introductory chapter we have outlined our requirements on the WSID system, of which the most important is that all steps of the method have to be performed in an unsupervised and knowledge-free manner. The motivation for this is that lexical resources and sense-annotated corpora are scarce, expensive, time-consuming to construct and often available only for limited number of most popular languages. Even though supervised approaches using such resources so far outperform their unsupervised counterparts, the ability to generalise unsupervised methods to different languages and domains is a strong motivation for continuation of research in this direction.

The first step in building such a system is to perform *word sense induction* (WSI), on which we focus in this chapter. Word sense induction is a problem concerned with automatic identification of the senses of a word from raw unstructured corpora. Therefore WSI approaches help to overcome the bottleneck of lexical resources and hand-annotated data, otherwise required in large quantities. As outlined in Section 2.2 of Related Work, WSI is usually performed with some form of clustering: either of contexts or of words.

In our work we show how already existing methods for training of word embeddings, measurement of word relatedness and clustering can be combined into a system producing senses vectors. Representing word senses with vectors, and not sense clusters or sense definitions, is desirable because they:

1. improve over single-prototyped word vectors by giving a true representation of polysemous words and open doors to new applications of word embeddings,
2. reduce complexity of WSD procedure with direct application of vector cosine similarity to identify correct meaning.

In this chapter we present our word sense induction method. In Section 4.1 we provide a high-level overview of the complete pipeline. In Sections 4.2 to 4.5 we go deeper into the details of every stage and describe how they work. Finally in Section 4.6 we illustrate the performance of our induction system and provide several examples of induced senses.

4.1 Overview

The goal of our WSI system is to induce senses from a raw corpus and to represent them with sense embeddings. In this work we use the words *embeddings* and *vectors* interchangeably. A sense embedding is a compact (several hundreds of dimensions) continuous-valued representation of a word meaning. Usually, to speak about semantics encoded in a word vector we examine which words are placed close to it in vector space (we call them *neighbours*, *similar words* or *related terms*). A sense vector corresponding to one of the meanings of a polysemous word, for example sense of $table_1$ as furniture, should have among its nearest neighbours only words like *chair*, *desk*, *counter* and nothing related to data.

There is no general answer to the question how many senses words have and where to draw a line between them. Usually linguists distinguish two types of sense ambiguity:

- **homonymy** – two completely unrelated senses. Examples of homonymy are *table* (furniture) / *table* (information), *mouse* (animal) / *mouse* (device), *bank* (finance) / *bank* (river). Even if some historical or symbolical link may be drawn between those words, they refer to different concepts, which are unlikely to appear in the same context.

- **polysemy** – two different but related to each other senses. Some examples include: *chicken* (animal) / chicken (food), *bank* (finance) / *bank* (building), *man* (species) / *man* (person). Usually polysemous words have a common origin.

In this work we do not differentiate between homonymy and polysemy. In fact, we refer to any ambiguous word as *polysemous*, and the right interpretation of this term in our context is *not monosemous*.

In Section 2.1 of Related Work we have shown that existing approaches for learning of sense embeddings (1) use context clustering, (2) directly learn sense embeddings with modified neural network architectures similar to Skip-Gram or (3) rely on sense inventories from lexical resources. In contrast to these techniques we propose to induce a sense inventory using existing word embeddings via clustering of ego-networks of related words.

Our method consists of four main stages depicted in Figure 4.1. First, (1) we learn word vectors with an existing system for single-prototype word embeddings. Then, (2) we use the vector space formed by these word vectors to calculate a word similarity graph, which contains all words in the vocabulary as nodes linked by edges weighted with similarity score between them. After that, (3) we extract ego-networks for words in the similarity graph and cluster those to induce senses. Finally, (4) we aggregate word vectors with respect to induced senses.

With respect to the set of requirements that we have outlined in Section 1.2 our WSI system:

- represents word senses with unique dense vectors,

- induces senses with a non-parametrized clustering algorithm and does not rely on any lexical resource apart from a raw text corpus,

- uses clustering algorithm that is known to produce coarse-grained sense distinction,

- integrates only on highly scalable existing methods that can handle corpora with over one million words in the vocabulary, thus ensuring the broad coverage of induced senses.

In addition to this, our system is very flexible. All stages of our method are independent of each other, and therefore, black boxes used to implement them are replaceable. Should a better system for learning of word embeddings or graph clustering appear, it can substitute the one used in our pipeline. The same independence of single stages means that our method can use existing word embeddings, word similarity graphs and sense inventories, which is very useful in situations when an external system is not publicly available or its output is not feasibly reproducible. For example, we can use an available sense inventory, thus skipping the stages two and three. Also we can work with alternative word similarity graphs (therefore, skipping the second stage). This opens up our system to a wide range of experimental scenarios, in which we can (1) evaluate various configurations of the system itself or (2) use it as framework for comparison of systems integrated into the pipeline in terms of their usefulness for word sense induction and disambiguation.

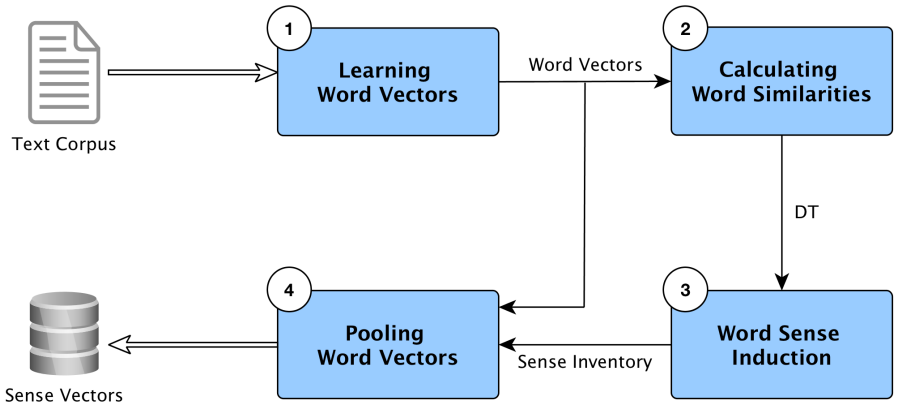


Figure 4.1: The complete word sense induction pipeline.

4.2 Learning Word Vectors

The goal of this stage is to train single-prototype word vectors from a large raw text corpus. This stage produces one vector for each uniquely spelled term in a corpus, conflating word senses if a word has many.

Word vectors are useful to us in many ways:

- The vector space provides us with a word similarity relation expressed as cosine similarity between vectors. We use it at second stage to build a word similarity graph.
- Word vectors can be combined using linear operations in a semantically meaningful way. We rely on this property at stage four, when we transform sense clusters into sense vectors.
- Word vectors can be used to represent context, in which a target word has to be disambiguated. We use them for this purpose in our WSD mechanism (Chapter 5)

To produce word vectors we first preprocess the text corpus and then run a word embeddings learning algorithm provided by *word2vec* toolkit. Our candidates for this step were Skip-gram/CBOW or GloVe word embedding models. Both have publicly available implementations. We have reported that they are currently considered to produce vectors of comparable quality [Levy et al., 2015]. We have decided to use *word2vec* because it has been introduced earlier, has been longer in use, and therefore, is easier to troubleshoot thanks to a larger user community.

For training, we use the original C implementation of *word2vec*, because it is significantly faster than the Python implementation distributed with *gensim* package [Řehůřek and Sojka, 2010].

Preprocessing

word2vec takes a tokenized text file as input. It interprets any sequence of chars surrounded by white spaces as a token. Separation of text into sentences or articles is irrelevant. The default preprocessing of *word2vec* consists of following steps:

- remove all URLs and tags,
- replace any character not in [a-z][A-Z][0-9] with a whitespace,

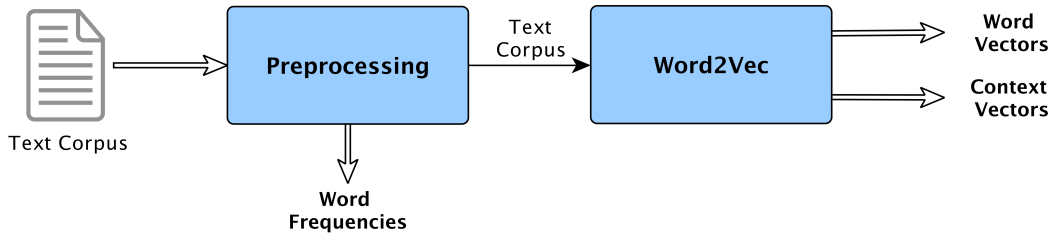


Figure 4.2: Stage 1: Learning word vectors.

- normalize each contiguous occurrence of white spaces to a single white space,
- lowercase all characters,
- spell out digits ("13" becomes "one three").

We consider this way of text cleaning too aggressive and simplifying. First of all, it splits hyphenated words like *mother-in-law* or *good-looking*, and as the result, removes them from model's vocabulary. Same thing happens to any word containing non-ASCII letters, i.e. *café*, *étude* or *crêpe*. Secondly, lowercasing all words may partially remove semantic information. For example, two meanings of the word *python* can be distinguished by the way the word is written in text: *python* (snake) is likely to start with a small letter, while *Python* (programming language) mostly appears capitalized. Finally, transforming numbers might also remove contextual information required during training. Consider two usages of the word *account*: (1) *transfer accounts to another branch* and (2) *accounts to 15%*. It is used as a *bank account* and as a verb respectively, and the number *15* is an important clue that emphasises this difference.

We propose to modify original corpora as little as possible. Our preprocessing algorithm:

- tokenizes text with Treebank tokenizer,
- converts to lower case only if users sets the respective parameter,
- leaves all characters intact.

For illustration we provide an example of text preprocessing of the sentence "*This state-of-the-art language model has been presented in Quebéc in 2009.*" produced by the default *word2vec* preprocessing script and our custom method:

default	this state of the art language model has been presented in qu bec in two zero zero six
custom	This state-of-the-art language model has been presented in Quebéc in 2009 .

In addition to preprocessing, at this stage we count word frequencies of the corpus. The output file consist of lines in the following format:

```
word<SPACE>count
```

where `count` is the number of times the `word` appears in the corpus. Each word is written on a new line and the list is sorted by the count number in decreasing order (most frequent terms are in the beginning). The file of word frequencies is not crucial for the next stages of the pipeline, but it may be used to prune words from the model or from a word similarity graph with respect to their frequency.

Training Word Vectors

After the corpus has been prepared, we can train word vectors with *word2vec*. Here we have to consider the importance of training parameters to our goal. Higher number of dimensions improves the quality of vectors. This is especially noticeable up to 200 hundred dimensions. On the other hand, it leads to a larger model and longer computation time of cosine similarity between vectors. The number of iterations can also positively influence vectors, however, it is more important if the corpus is relatively small (several gigabytes or less). Setting the minimum word frequency higher will decrease the vocabulary size of the model and may change the list of nearest neighbours extracted for words. Other parameters include the size of a context window and type of the model (CBOW/Skip-gram), but we cannot foresee their influence on the performance of our system without experiments.

We have modified the original implementation of *word2vec* to save both weight matrices trained by the algorithm. Independently of the model type, *word2vec* saves only the matrix $\mathbf{W}_{V\times N}$ (see architecture of models depicted in Figure 3.1) and suggests to use it as word vectors. We also save the complementary matrix $\mathbf{W}'_{N\times V}$ and refer to it as *context vectors*. We use them later in one of our WSD mechanism. These vectors are not strictly necessary for the next stages of word sense induction.

All vectors can be saved in a plain text format or binary format. Vectors are ordered by the frequency of words they represent (vectors of most frequent words come first). Loading these vectors with *gensim* package preserves this order.

4.3 Calculating Word Similarity Graph

The goal of this stage is to calculate the word similarity graph. This graph contains all words of the vocabulary as nodes linked by edges weighted with the similarity score between them. The graph is undirected.

As shown in Figure 4.3, to build this graph we propose to use two different similarity measures introduced in Section 3.2. The first is based on the word embeddings trained at the previous stage (sim_{w2v}) and the second one is based on word features approach provided by *JoBimText* framework (sim_{JBT}). Multiple alternatives exist for computation of word similarities [Zhang et al., 2013]. We have decided for those two because (1) both are scalable and allow to build a graph for all words in the vocabulary; (2) they are interesting to compare, being based on two different types of context observations. While *word2vec* interprets context as a bag of words, *JoBimText* works with syntactical dependencies.

It is important to clarify that the notion of similarity implemented by those measures is much wider than word synonymity and not limited by the part of speech of words. For example, neighbours of the word *warm* may include terms like *hot*, *cold* and *temperature*. In literature this relation is also called *word relatedness*, meaning that it measures word topical similarity and includes, but is not limited to, synonyms, antonyms, hypernyms, hyponyms, etc. This is not an obstacle for word sense induction. In fact, two senses *hot* (temperature) and *hot* (attractive) may be well distin-

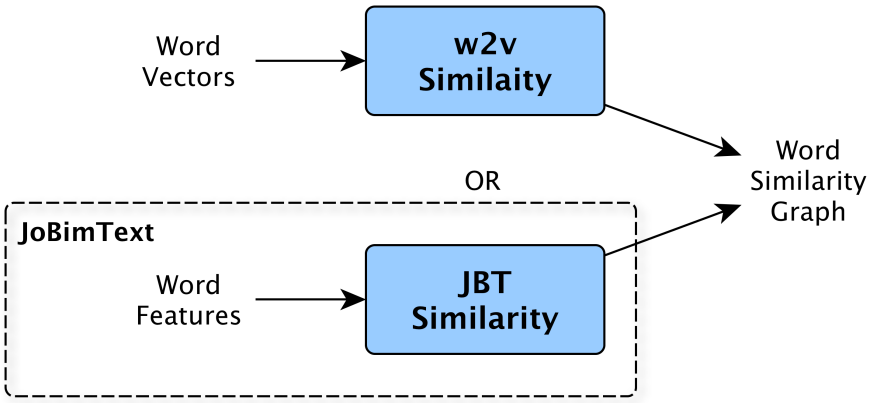


Figure 4.3: Stage 2: Calculating word similarity graph.

guished with two groups of its synonyms, antonyms and other semantically related words: {*cold*, *warm*, *temperature*, *weather*} and {*good-looking*, *ugly*, *interesting*, *boring*} respectively.

The output of this stage is a word similarity graph. We represent it in form of *distributional thesaurus* (DT). A DT file consists of lines in the following format:

`word1<TAB>word2<TAB>score`

where `score` is the similarity score between `word1` and `word2`. Thus, each line represents an edge of the graph. Lines in DT are usually grouped by the first word and therefore, DT can also be seen as a dictionary of related terms.

The graph produced at this stage is not fully connected. For each word of the vocabulary included in the graph we draw at most N edges to its most similar terms. This is motivated by prior studies [Biemann et al., 2013, Panchenko, 2013]: as observed, words have only several hundred strongly semantically related words. In fact, we usually set N around 200.

Word2vec similarity

In this case, the similarity between two words is defined as a cosine similarity of their vectors:

$$sim_{w2v}(w_i, w_j) = \frac{\mathbf{w}_i \cdot \mathbf{w}_j}{||\mathbf{w}_i|| \cdot ||\mathbf{w}_j||}, \tag{4.1}$$

where $\mathbf{w}_i$ and $\mathbf{w}_j$ are word embeddings of words w_i and w_j respectively. To build a graph we iterate over all words in the vocabulary of the model trained at the first stage. For each word we extract N most similar terms. To find those most similar terms we have to calculate similarities to all words in the vocabulary and sort them by the score. To do so based on a model that includes more than one million words is very time consuming. To speed up this process we perform block-wise matrix multiplication.

Let $\mathbf{M}_{V \times D}$ be the input matrix of normalized word embeddings, where V is the size of the vocabulary and D is the dimensionality of vectors. We calculate word similarities for a block of B words at a time. Let $\mathbf{M}_{B \times D}$ be a matrix of B normalized word embeddings belonging to the block. We compute the product of these matrices:

$$\mathbf{M}_{B \times V} = \mathbf{M}_{B \times D} \cdot \mathbf{M}^{\intercal}_{V \times D}, \tag{4.2}$$

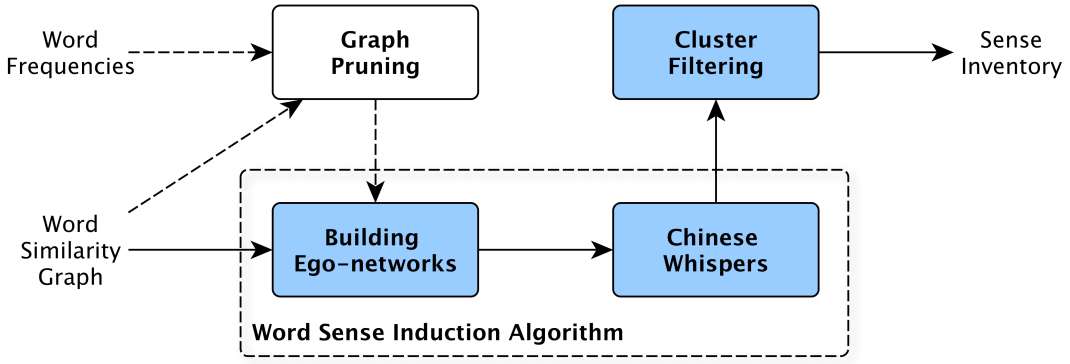


Figure 4.4: Stage 3: Word sense induction.

where an element m_{ij} of the matrix $\mathbf{M}_{B\times V}$ corresponds to cosine similarity between vectors $\mathbf{w}_i$ and $\mathbf{w}_j$.

At this stage there is a possibility to filter words in the vocabulary. We propose to calculate neighbours only for words that include no other chars rather than upper and lower case letters, a point and a dash. This operation also preserves the original word order in the vector model produced by *word2vec*.

JoBimText similarity

In this case we use the similarity measure implemented in JoBimText framework. The similarity between two words is defined as the number of features they share:

$$sim_{JBT}(w_i, w_j) = features(w_i) \cap features(w_j), \tag{4.3}$$

where features are observations of words in context, for example observed dependency relations, normalized and pruned down to a recommended number of features per word. For computation of a word similarity graph using this measure we rely on JoBimText implementation. In this framework computation of a DT is one of the intermediate steps. More information about the framework and how to use it can be found at its homepage[1].

4.4 Word Sense Induction

The goal of this stage is to produce a sense inventory. In such inventory a word sense is represented by a word cluster. For instance the cluster "*chair*, *bed*, *bench*, *stool*, *sofa*, *desk*, *cabinet*" can represent the sense *table* (furniture).

We achieve this goal via clustering of word graphs, similar to [Pantel and Lin, 2002] and [Biemann,]. In particular, to induce senses of a word t we first construct an ego-network G of this word and then perform graph clustering of this network. The identified clusters are interpreted as senses. An ego-network consists of a single node (ego) together with the nodes it is connected to (alters) and all the edges among those alters. An ego-network of size 20 for a word *mouse* is shown in Figure 4.5 as an example. All information necessary to build an ego-network for all words in the vocabulary is already available to us in form of a DT built at the second stage.

[1] `http://maggie.lt.informatik.tu-darmstadt.de/jobimtext/`

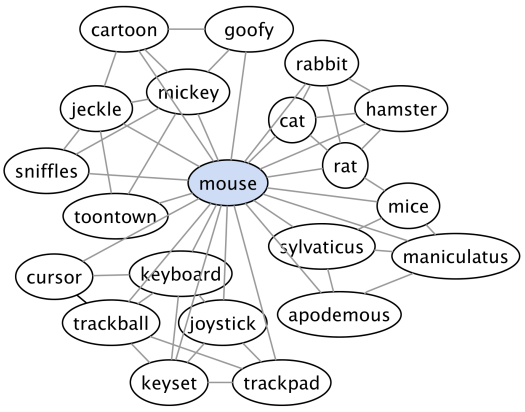


Figure 4.5: An ego-network of size 20 for a word *mouse*.

The success of our approach is based on two hypotheses:

1. an ego-network of a limited size of a polysemous word includes words related to its different senses,

2. words related to the same sense are tightly interconnected while having fewer connections to words referring to other senses.

Our exemplary ego-network of a word *mouse* complies with these hypotheses.

Method description

As depicted in Figure 4.4, this stage consists of three main steps: (1) pruning of a word similarity graph, (2) word sense induction, separated into building of ego-networks and their subsequent clustering with Chinese Whispers algorithm and (3) filtering of clusters.

First step, pruning of the word similarity graph, is optional. In some situations we might want to decrease the size of this graph, either to speed up the clustering or to reduce the memory load. We suggest to do this with respect to word frequencies, assuming that sense induction might not be relevant for very infrequent terms, which, however, contribute significantly to the graph's size. We can delete word pairs from the distributional thesaurus with help of word frequencies of the corpus counted during preprocessing at the first stage. A pair is removed, if any of its words has a count lower than a set threshold.

The second step is our word sense induction method, presented in Algorithm 2. It processes one word t of the word similarity graph T per iteration. First, we collect N most similar terms of t in a set denoted as V. These are the nodes of the ego-network G, which so far has no edges. After that we iterate over nodes v in V. For each node we collect n of it most similar terms. If a new similar term is already present in V, we draw an edge from it to v. Otherwise we do nothing. This way we build an ego-network G. Note that it does not contain the ego word t, and therefore, has no edges from t to its similar terms. In the end, we pass the ego-network G to the Chinese Whispers algorithm and it returns sense clusters of the word t.

We have considered two candidates for the clustering step: Markov cluster algorithm (MCL) and Chinese Whispers (CW). Both are parameter-free clustering algorithms which discover the number of sense themselves. However, we have decided for CW because (1) it is a time-linear algorithm

that works faster than MCL in our scenario and (2) it does not require an additional granularity parameter. Instead, the granularity of clusters is governed by the structure of an ego-network. In addition to this, CW showed promising sense induction results in the past [Biemann,].

Algorithm 2: Word sense induction.

input : T – word similarity graph, N – ego-network size, n – ego-network connectivity
output: for each term $t \in T$, a clustering S_t of its N most similar terms

```
foreach t ∈ T do
    V ← N most similar terms of t from T
    G ← graph with V as nodes and no edges E
    foreach v ∈ V do
        V' ← n most similar terms of v from T
        foreach v' ∈ V' do
            if v' ∈ V then add edge (v, v') to E
        end
    end
    S_t ← ChineseWhispers(G)
end
```

Our induction algorithm has two parameters: size of the ego-network N and connectivity of the ego-network n. With N we can increase the number of similar terms collected for each word and in doing so, increase the chance of getting terms related to different senses. With n we can control the granularity of clusters. n is a maximum number of edges a node can have in the network. By decreasing it we decrease the number of edges in a network and in doing so, decrease the chances of CW to merge two clusters together, thus producing finer-grained clusters. In Figure 4.6 we show variations in the clustering of an ego-network of the word *mouse* built with fixed $N = 20$, and varied n. As we can see, CW outputs three clusters with $n = 10$ and four clusters with $n = 5$.

At the final step of this stage we filter sense clusters. The goal of filtering is to remove very small clusters which are usually the result of noise in an ego-network. We remove any cluster of size less than a parameter k. This parameter should be set with respect to the size of the ego-network N, for example $k \in [5, 15]$ can be considered for $N = 200$.

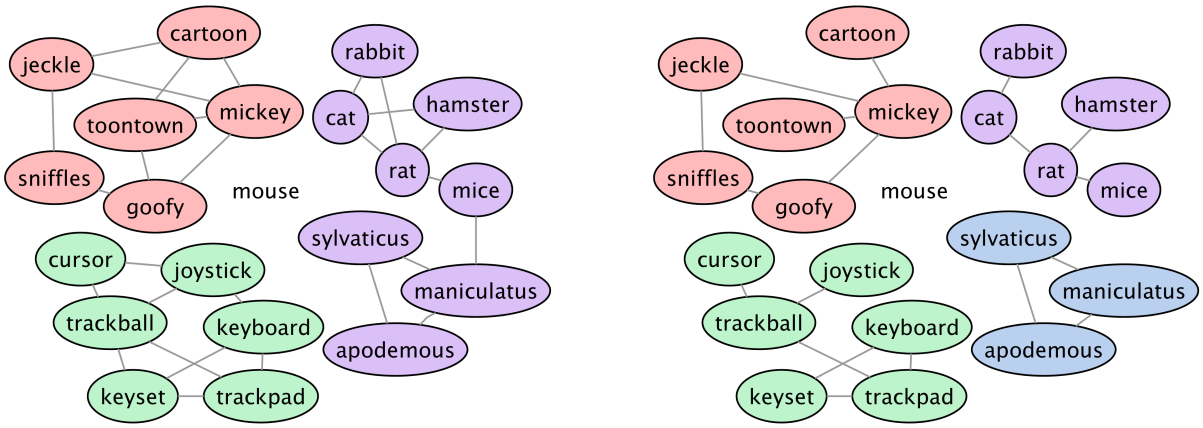
The output of this stage is a sense inventory. It contains senses of all words in a word similarity graph, unless some of the words where pruned before clustering. Each sense of a word is represented with a set of its related terms (sense cluster). Each line in the output file corresponds to one word sense written in the following format:

`word<TAB>sense_id<TAB>rel_terms`

where `sense_id` is an integer identifier of a sense of the `word`. This identifier is unique in scope of the respective `word`. `rel_terms` is a sense cluster written in the following format:

$\texttt{term}_1\texttt{:weight}_1\texttt{,term}_2\texttt{:weight}_2\texttt{,...}$

where $\texttt{term}_\texttt{i}$ is an $\texttt{i}^{\texttt{th}}$ word included in the cluster and $\texttt{weight}_\texttt{i}$ is the weight of the edge between $\texttt{term}_\texttt{i}$ and the `word` taken directly from the word similarity graph. In case of a word similarity graph built with *word2vec* it is a float number that equals to cosine similarity between respective



(a) $N = 20$, $n = 10$ **(b)** $N = 20$, $n = 5$

Figure 4.6: Clusters produced by Chinese Whispers of an ego-network for the word *mouse* with varied parameter n. Note that the ego word *mouse* is excluded from clustering.

word vectors. If the graph was built with *JoBimText*, it is an integer number corresponding to the number of shared features. All terms in the `rel_term` list are listed in decreasing order of the `weight` value.

4.5 Pooling of Word Vectors

The goal of this stage is to transform sense clusters into sense vectors. We achieve this by averaging vector representations of words in a cluster. Our idea relies on the additive compositionality property of word vectors discussed in Section 3.1. As it has been shown in one of the examples, the element-wise addition of vectors representing words *Germany* and *river* conflated meaning of these concepts and returned a vector close to those of German rivers (*Elba*, *Rhine*, *Danube*).

As shown in Figure 4.7, this stage requires a sense inventory and word vectors as input. We iterate over the sense inventory and produce a sense vector for each sense cluster. We define a sense vector as a function of word vectors representing items of a cluster.

Let W be a set of all words included in the word vector model and let $S_i = \{w_1, \ldots, w_n\} \subseteq W$ be a sense cluster from the sense inventory obtained at the previous stage. Consider a function $vec_w : W \rightarrow \mathbb{R}^m$ that maps words to word vectors and a function $\gamma_i : W \rightarrow \mathbb{R}$ that maps words from the cluster S_i to their weights. We propose two ways to calculate sense vectors:

unweighted average of word vectors:

$$\mathbf{s}_i = \frac{\sum_{k=1}^{n} vec_w(w_k)}{n};$$

and weighted average of word vectors:

$$\mathbf{s}_i = \frac{\sum_{k=1}^{n} \gamma_i(w_k) vec_w(w_k)}{\sum_{k=1}^{n} \gamma_i(w_k)}.$$

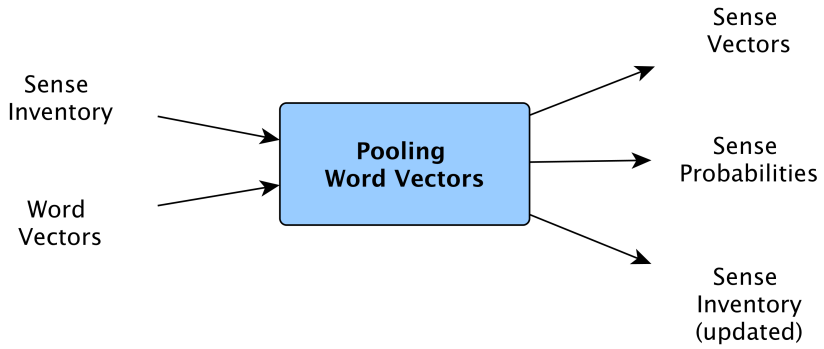


Figure 4.7: Stage 4: Pooling of word vectors.

This stage outputs (1) sense vectors, (2) sense probabilities and (3) an updated sense inventory. Sense vectors are saved in the *word2vec* format, with each word sense identified as a concatenation of the `word` and a `sense_id` in the following format:

```
word#sense_id
```

Sense vectors can therefore be loaded with original *word2vec* toolkit or *gensim* package.

During the pooling of word vectors we also calculate probabilities of senses. We define it with respect to the size of a sense cluster. Consider a set of sense clusters $S^w = \{S_1^w, \ldots, S_n^w\}$ of a word w. A probability of a sense S_i^w is defined as:

$$P(S_i^w) = \frac{|S_i^w|}{\sum_{j=1}^{n} |S_j^w|}.$$

This definition is only an approximation of the real sense probability which could be calculated on the original corpus with respect to the number of occurrences of each sense. However, we do not have a possibility to compute it this way and instead we assume that senses, which are more frequent, will also have more related terms.

Finally, we explain why we make an update of a sense inventory. In Overview of our WSI method (Section 4.1) we have listed flexibility as one of its properties. Indeed, at this stage we can use any external sense inventory that represents senses with word clusters as input. In rare cases, the vocabulary of the sense inventory provided as input might not be completely covered by the word vector model. Therefore, some words in a cluster or complete word senses might be ignored during pooling of vectors. Depending on a sense inventory type and whether it uses word collocations (for example *female actor* as a related term of *actress*), user can decide if collocations should be split or not. An updated sense inventory reflects these changes and contains only elements that were in fact used for computation of sense vectors.

4.6 Examples

In this section we provide examples of word senses induced with our method, as well as results of intermediary stages.

table		mouse		ruby	
1 – 10	100 – 110	1 – 10	100 – 110	1 – 10	100 – 110
tables	desks	mickey	typewriter	sapphire	jewel
lookup	queue	mice	snoopy	rubies	judy
lists	pointers	cat	duck	jruby	cabochon
sortable	asterisked	apodemus	bug	macruby	rexx
diagram	calcrete	toontown	mousepad	matz	megan
rows	menu	rabbit	click	diamond	mixins
parentheses	column	trackball	tigger	keeler	haskell
tabular	dime	sniffles	screensaver	rubinius	violet
lists	summary	jeckle	keystroke	spinel	trixie
brackets	sofa	rat	animation	realbasic	billy

Table 4.1: Nearest neighbours of words *table*, *mouse* and *ruby*. First ten neighbours from the first hundredth and from the second hundredth for each word. Neighbours extracted with *word2vec* similarity from vectors trained on 1.5 GB excerpt of an English Wikipedia dump.

In the design of our method we have made several theoretical assumptions about the word's ego-networks and their usefulness for word sense induction. To understand if these assumptions in fact hold in practice we conducted a small preliminary experiment. It provides evidence that our method is indeed able to produce meaningful sense vectors.

Setup

In this experiment we have induced senses from an excerpt of an English Wikipedia dump. The corpus was 1.5 GB in size and cleaned with *word2vec* default preprocessing script. We trained a CBOW model with word vectors of 200 dimensions using a context window size 8 and performing 15 training iterations.

Stage 2: Word neighbours

First we observe the nearest neighbours of several polysemous words to see if they indeed include terms related to different senses. We extract 200 nearest neighbours using sim_{w2v} similarity measure. The results for words *mouse*, *table* and *ruby* are given in Table 4.1. For each word we display first ten neighbours from the first hundredth and from the second hundredth. We observe that neighbours in fact refer to different senses, for example the word *mouse* is close to *mickey*, *rat* and *trackball*. Sometimes one sense clearly outweighs the other, like it does for *table*. It has more neighbours related to the *data* sense than to *furniture*.

Stage 3: Sense clusters

After that we cluster word similarity graph with Chinese Whispers using parameters $N = 200$, $n = 200$ and filtering out all clusters with less than 5 elements. Senses that our model is able to induce are given in Table 4.2. This confirms our hypothesis, that in an ego-network terms related to one sense are stronger interconnected than terms related to different senses. We also observe

Word	Id	Cluster size	Top cluster words
table	0	165	tables, lookup, lists, sortable, diagram, rows, parentheses
table	1	34	tray, drawers, trays, bucket, meishi, pigeonholes, lectern, billiard
mouse	0	43	mice, apodemus, rat, sylvaticus, peromyscus, maniculatus, hamster
mouse	1	120	mickey, cat, toontown, rabbit, sniffles, jeckle, cartoon
mouse	2	37	trackball, mouses, keyset, keyboard, chording, cursor, trackpad
ruby	0	53	jruby, macruby, matz, rubinius, realbasic, ironruby, vroom, perl
ruby	1	73	sapphire, rubies, diamond, spinel, emerald, opal, padparadscha
ruby	2	72	keeler, macie, vicki, stuntz, eloise, jack, kelly, lucy, gayle, tiffany

Table 4.2: Sense clusters of words *table*, *mouse* and *ruby*. Weights of cluster words are omitted. Senses induced with *word2vec* similarity from vectors trained on 1.5 GB excerpt of an English Wikipedia dump.

that cluster sizes differ significantly. Note that our model does not differentiate between lower and upper case letters, therefore a sense of the word *Ruby* related to female names is also included.

Stage 4: Sense vectors

Finally, we want to understand how does our approach for calculation of sense vectors from sense clusters work. We have computed sense vectors using unweighted pooling of word vectors in a sense cluster. Nearest neighbours of all senses of words *table*, *mouse* and *ruby* are listed in Table 4.3. We observe that while nearest neighbours of sense vectors are not identical to the words in the corresponding sense cluster, they are semantically similar and sufficiently sense-specific.

In this section we have cherry-picked examples of successful performance of our word sense induction system. Obviously, there are also situations when it does not perform as expected. Such cases will be discussed in more detail in Section 6.4.

Sense	Interpr.	Top neighbours
table#0	data	sorted#0, tables#1, leftmost#0, tabular#1, concatenating#0
table#1	furniture	tables#0, drawers#0, chairs#1, billiard#1, desk#0, abattant#0
mouse#0	animal	falettinme#1, rat#0, mouses#1, hamster#0, nutcracker#1, dormouse#0
mouse#1	cartoon	horsecollar#1, hippety#1, clarabelle#0, mickey#0, plucky#1, sniffles#0
mouse#2	device	keyboard#1, chording#1, typeball#0, snood#3, spacebar#0, chorded#1
ruby#0	language	groovy#1, generics#0, php#0, jruby#0, gobject#0, actionscript#0
ruby#1	gemstone	emerald#0, moonstone#1, emeralds#0, gems#0, tlich#1, diamond#0
ruby#2	name	connie#0, skeeter#0, patsy#0, gayle#0, patty#0, loretta#0

Table 4.3: Nearest neighbours of sense vectors for words *table*, *mouse* and *ruby*. Sense vectors calculated from sense inventory induced with *word2vec* similarity from vectors trained on 1.5 GB excerpt of an English Wikipedia dump.

5 Word Sense Disambiguation

In the previous chapter we have proposed a knowledge-free system for induction of word senses from raw text corpora. In this chapter we describe a word sense disambiguation mechanism built on top of it.

The task of *word sense disambiguation* is to automatically identify which sense of a word is used in a sentence, when a word has multiple meanings. This task can be solved for all words in a sentence (*all-words WSD*) or just for one target word in a sentence (*lexical sample WSD*). In Section 2.2 we have outlined three groups of approaches used to solve the WSD problem: knowledge-based, supervised and unsupervised.

Our system falls into the third category. To solve the task it requires only word embeddings, sense embeddings, and optionally, context word embeddings. First of all, the number of senses for each word is defined by the induced sense inventory and is therefore not bound to any existing knowledge base or lexical resource. Secondly, the semantic information encoded in word, sense and context embeddings is the only source of disambiguation clues. In compliance with our requirements this mechanism is fully unsupervised and knowledge-free.

5.1 Method

The WSD problem can be formally defined as follows: given a word w and a set of its context words C the goal is to define a mapping $D(w, C) \in S^w$, where S^w is a set of senses of the word w according to some sense inventory. Our method consists of three stages: (1) context extraction, (2) context filtering and (3) disambiguation according to one of two disambiguation strategies. It takes a target word and the sentence in which it appears as input, and outputs a sense identifier. To operate it needs (1) a collection of sense vectors, which acts as a sense inventory, and (2) either word vectors or context vectors, depending on the disambiguation strategy.

Context extraction

Given a sentence and a target word w we tokenize the sentence and extract a fixed number of tokens on the left and on the right of the target word (this number is called *context window size*). We refer to these tokens as the context $C = \{c_1, \ldots, c_k\}$ of w. No removal of stop words or punctuation happens at this step. The order of context words is irrelevant.

Disambiguation strategies

Given a target word w and its extracted context C we first map w to a set of its sense vectors $S^w = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ available in the provided collection of sense vectors. We define a function $vec_w : W \rightarrow \mathbb{R}^m$ that maps words to word vectors and a function $vec_c : W \rightarrow \mathbb{R}^m$ that maps words to context vectors. For explanation of the difference between word vectors and context vectors refer to Section 4.2. We use two strategies to choose a correct sense in context.

The first disambiguation strategy is based on the probability of a sense in the given context. The roots of this idea lie in the architecture of *word2vec* language models that learn vectors by trying

to predict the word given its context or vice versa, depending on the model type. In this strategy we use context vectors to represent words in C. The mapping $D_{prob}(w, C)$ is defined as:

$$D_{prob}(w, C) = \underset{\mathbf{s}_i \in S^w}{\operatorname{argmax}} P(C|\mathbf{s}_i) = \underset{\mathbf{s}_i \in S^w}{\operatorname{argmax}} \frac{1}{1 + e^{-\bar{\mathbf{c}}_c \cdot \mathbf{s}_i}}, \tag{5.1}$$

where $\bar{\mathbf{c}}_c$ is the mean of context vectors representing context words in C:

$$\bar{\mathbf{c}}_c = \frac{\sum_{i=1}^{k} vec_c(c_i)}{k}.$$

The second disambiguation strategy is based on similarity between senses and context. It is inspired by the ability of word embeddings to place vectors of semantically similar words closer together. For this strategy we use word vectors to represent words in C. This is practical, as the standard implementation of *word2vec* does not save context embeddings and thus most pre-computed models provide only word vectors. The mapping $D_{sim}(w, C)$ is defined as:

$$D_{sim}(w, C) = \underset{\mathbf{s}_i \in S^w}{\operatorname{argmax}} sim(\mathbf{s}_i, C) = \underset{\mathbf{s}_i \in S^w}{\operatorname{argmax}} \frac{\bar{\mathbf{c}}_w \cdot \mathbf{s}_i}{||\bar{\mathbf{c}}_w|| \cdot ||\mathbf{s}_i||}, \tag{5.2}$$

where $\bar{\mathbf{c}}_w$ is the mean of word vectors representing context words in C:

$$\bar{\mathbf{c}}_w = \frac{\sum_{i=1}^{k} vec_w(c_i)}{k}.$$

Context filtering

To improve WSD performance we also apply context filtering. Typically, only several words in context are relevant for sense disambiguation, like *chair* and *kitchen* are for *table* in *"They bought a table and chairs for kitchen."* Using all words from the context window, including articles, pronouns and other non-discriminative words, smoothes sense-related traits of the context and hinders disambiguation of the target word. Therefore, the idea is to use only several most discriminative words from the context. For each word c_j in context $C = \{c_1, \ldots, c_k\}$ we calculate a score that quantifies how well it discriminates the senses. The score of a context word c_j is computed as follows:

$$\max_i f(\mathbf{s}_u, c_j) - \min_i f(\mathbf{s}_v, c_j), \tag{5.3}$$

where $\mathbf{s}_u$ and $\mathbf{s}_v$ iterate over senses of the ambiguous word and f is one of our disambiguation strategies: either $P(c|\mathbf{s})$ or $sim(\mathbf{s}, c)$. The parameter p defines how many most discriminative context words will be used for disambiguation.

5.2 Example

In this section we illustrate three stages of our WSD system. We want to disambiguate the word *table* in the sentence *"They bought a table and chairs for kitchen."* For this example we used the same word and sense vectors that were obtained during the preliminary experiment on word sense induction described in Section 4.6. We perform disambiguation using similarity disambiguation

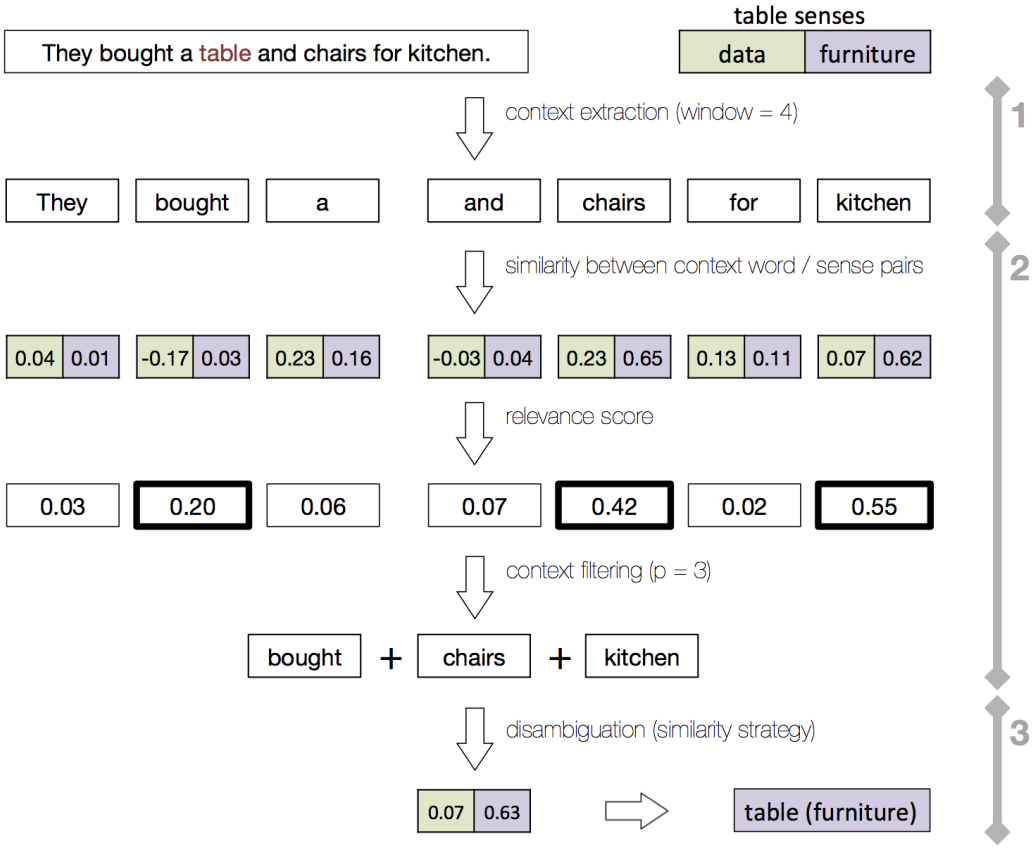


Figure 5.1: Three stages of word sense disambiguation of the word *table* in the sentence *"They bought a table and chairs for kitchen."*: (1) context extraction with window set to 4, (2) context filtering with threshold p set to 3 and (3) disambiguation using similarity strategy.

strategy, context window of size 4 and filtering of contexts with parameter p set to 3. This setting of parameters was chosen arbitrarily and was not governed by any evidence of good performance.

The actions performed by the WSD mechanism during disambiguation are depicted in Figure 5.1. Our sense vectors contain two senses for the word *table*: one related to *data* and another related to *furniture*. First we tokenize the sentence and extract four words on the left of the target word and four on the right. Because the sentence has only three words on the left, we end up with 7 context words.

Next three steps handle the context filtering. First, we calculate the similarity between each separate context word c_j and each sense s_i of the target word using the similarity strategy $sim(\mathbf{s}_i, c_j)$. We can observe that words like *a* and *for* are almost equivalently similar to both senses, while *chair* and *kitchen* clearly tend toward the *furniture* sense. We calculate the relevance score as defined in Equation 5.3 and choose the three most informative words.

Finally, we perform the disambiguation with similarity strategy (see Equation 5.2) using an average vector of three chosen context words. Our model correctly identifies the meaning of the word *table* to be related to *furniture*. The margin between two sense scores is large: 0.63 vs. 0.07,

in part thanks to the context filtering. Without it, the sense scores are 0.54 and 0.15 respectively and the margin is smaller. While this causes no problem in this particular example, enlarging the margin between sense scores increases chances of a correct sense choice in less trivial situations.

6 Evaluation on TWSI

This chapter is dedicated to the evaluation of our system on word sense disambiguation task using the Turk Bootstrap Word Sense Inventory (TWSI) 2.0 dataset. TWSI is a large collection of sentences with a single sense-annotated word with relation to an induced sense inventory based on a lexical substitution task. The main goal of experiments described here is to understand the role and influence of our system parameters, and possibly find a best performing configuration that could act as default.

This chapter is structured as follows: In the first section we discuss alternatives for the evaluation of our system. Then, in Section 6.2 we introduce the TWSI dataset, evaluation metrics and baselines. Section 6.3 contains results of our experiments and their interpretation. Finally, in Section 6.4 we perform error analysis to better understand the flaws of our system.

6.1 Evaluation Alternatives

In this work we develop a word sense induction and disambiguation system, in which senses are represented with vectors. This opens up a wide range of evaluation possibilities.

On the one hand, we can follow intrinsic evaluation approaches used in many works related to development of multi-prototype word embeddings (see Section 2.1). Those test the quality of produced vectors on word similarity tasks. The idea behind these experiments is that multi-sense representation of words are expected to improve over single-prototype embeddings in cases when at least one element of a word pair is polysemous. If a model can differentiate between two senses of the word *bank* (*finance* and *riverbank*), it assigns a higher similarity score to the pair *bank/money* by choosing a sense vector for *bank* that maximizes pair similarity. In contrast, a model that conflates all senses in one word vector will deliver a lower score. This type of evaluation has been used in [Reisinger and Mooney, 2010, Huang et al., 2012, Tian et al., 2014, Neelakantan et al., 2014, Li and Jurafsky, 2015, Bartunov et al., 2015, Rothe and Schütze, 2015]. Among the most often used datasets for this task are WordSim353 [Finkelstein et al., 2002] and SCWS [Huang et al., 2012].

On the other hand, we could focus on extrinsic evaluation, by applying our system to word sense induction and disambiguation tasks directly. In these, the model has to predict a correct sense of a word in context and the number of correct predictions evaluates performance of both induction and disambiguation steps. Similar to the decision of the authors of the AdaGram model for sense embeddings [Bartunov et al., 2015], we have decided that this is a more suitable evaluation approach for our system. It has been used in [Bartunov et al., 2015] and [Rothe and Schütze, 2015]. It is usually performed on one of the datasets of Semantic Evaluation competition. We have decided to make two series of experiments: one on the TWSI dataset for parameter tuning (to which this chapter is dedicated) and another one on SemEval-2013 Task 13 dataset [Jurgens and Klapaftis, 2013] for comparison with other approaches (described in the next chapter).

Word	Substitutions
folk	ethnic:12, folk genre:8, country:8, simple style:7, folk music:6, traditional:4, ...
folk	people:9, person:4, group:2, personnel:2, individual:2, community:1, leader:1, ...
magazine	publication:42, periodical:32, journal:30, manual:9, gazette:5, newsletter:4, ...
magazine	cartridge:6, clip:5, chamber:3, holder:3, mag:3, cache:2, loading chamber:2, ...
tour	circuit:18, road show:11, trip:10, journey:10, concert tour:9, expedition:8, show:7, ...
tour	journey:10, visit:10, expedition:10, trip:9, travel:5, sightsee tour:4, route:4, ...
tour	assignment:5, voyage:5, stint:5, tour of duty:4, circuit:4, duty:4, bout:4, ...

Table 6.1: Examples of the TWSI inventory entries for several polysemous words.

6.2 TWSI Dataset and Evaluation Metrics

Original dataset

TWSI 2.0 [Biemann, 2012] is a coarse-grained sense inventory for lexical substitution of 1,012 highly frequent English nouns in Wikipedia. Each sense is represented with a list of words that can substitute the target noun in a given sentence. It is accompanied by a large collection of sentences with sense-annotated occurrences of target words.

This lexical resource was created by a relatively low-cost crowdsourcing process using Amazon Mechanical Turk[1]. Workers were asked to supply a list of substitutions for a word in its sentence-wide context. All sentences were taken from English Wikipedia dump of January 2008. To induce senses, the occurrences of the same word were automatically clustered by the overlap of their substitutions. Further tasks included manual verification of clustering and sense annotation of target words in a collection of sentences. Table 6.1 provides senses of exemplary polysemous words in the resulting sense inventory. Substitutions are weighted by the number of times they were provided by the annotators for a target in context. We can observe that for some words (*folk, magazine*) senses are very distinct, while in case of the *tour* the difference in meanings is quite subtle. Substitutions are provided in form of words or word collocations. On average, each sense is represented with 8-9 substitutions.

TWSI sense inventory covers 1,012 words, split into 2,443 senses with an average of 2.41 senses per word. The average number of senses is about a third compared to WordNet, and similar to other coarse-grained inventories such as OntoNotes [Hovy et al., 2006]. Figure 6.1 demonstrates that a majority of words have from 2 to 6 senses, but more than a third are monosemous. The dataset of sentences with sense-annotated target word occurrences has 145,140 instances, with almost 79% of contexts assigned to the most frequent senses.

Sense-balanced dataset

As we can see from the statistics above, the TWSI dataset is skewed towards the most frequent sense and includes a significant amount of sentences with monosemous target words. While this honestly reflects the sense distribution in real-word language usage and is undoubtedly a relevant test set for evaluation of a WSID system, we have decided to create an additional sense-balanced subset of TWSI with an equal number of sentences for each sense and including only ambiguous

[1] http://www.mturk.com

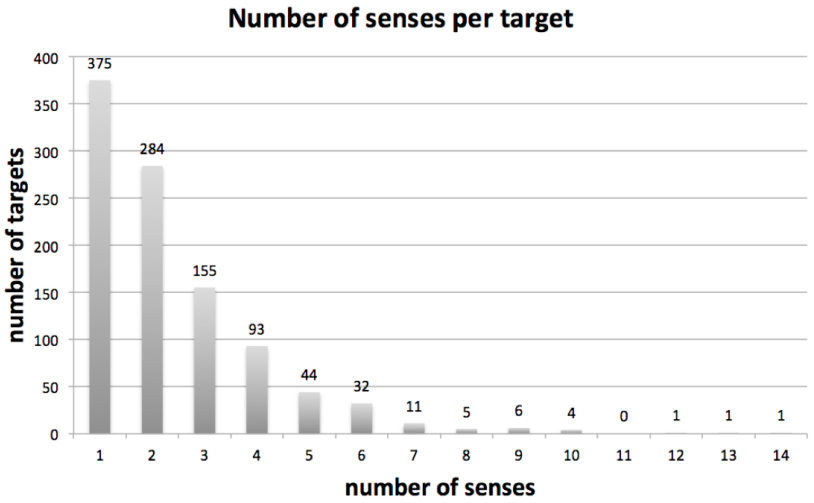


Figure 6.1: Distribution of number of senses per target in the unbalanced TWSI dataset.[2]

target words. It will ensure a better interpretability of the results which is especially important at the early stages of parameters tuning. We perform the following steps:

1. remove all monosemous words,
2. remove a word if any of its senses has less than five sentences in the dataset,
3. for all remaining words sample five random sentences from the original dataset.

This results in a small development set with 463 words and 6,165 test instances. In comparison to the original dataset it covers 45% of nouns and 4.25% of test instances. The average number of senses increased only slightly (from 2.41 to 2.66), from which we draw a conclusion that together with monosemous words we also removed most of the words with very fine-grained sense distinction. In the remaining of this thesis we refer to the original dataset as *full* or *unbalanced* TWSI and to the balanced version as *balanced* TWSI.

Mapping of inventories

One common problem in the evaluation of WSID systems is how to compare predicted sense labels from an induced sense inventory with golden labels of a lexical resource. Obviously senses of two inventories may differ in labels or sense distinction and a straightforward comparison is not possible. Therefore, it is necessary to perform a mapping from induced senses to the dataset senses. Methods proposed to solve this problem fall into two categories [Manandhar and Klapaftis, 2009]:

- *supervised evaluation*, when the first part of predictions is used to learn the mapping, and the second part is used to evaluate WSD performance.
- *unsupervised evaluation*, where sense disambiguation is regarded as a clustering task and evaluation is performed with measures of clustering quality.

[2] The figure taken from [Biemann, 2012].

We describe several of this metrics in detail in Section 7.1.

Proposed metrics are designed to handle the mapping problem in a situation, when sense labels are the only available information. However, in our scenario we have human-readable sense inventories for both the TWSI dataset and for senses produced by our system. Each sense is represented with a set of substitutions or related terms respectively. We propose a direct mapping approach based on cluster overlap.

We define the mapping as a partial function $mapping : U \mapsto T \cup \{\texttt{None}\}$, that maps each sense from the user sense inventory U to at most one sense from the TWSI inventory T. The mapping algorithm is presented in Algorithm 3. For each user word sense we assign the closest TWSI sense with respect to the measure *cosine_sim* of their clusters. This measure calculates cosine similarity for vectors of word weights, regarding only words that appear both in the user cluster and the TWSI cluster. If clusters have no common words, then the similarity equals zero. If it equals zero for all TWSI senses of the same target word, then a user sense has no match in T and the mapping function for this sense is undefined (`None`).

Please note that *mapping* is neither injective, nor surjective. Several custom senses may be mapped to the same TWSI sense; usually it happens if meanings of those custom senses are very close. Also, not all TWSI senses get a pair, which is a sign that a particular TWSI sense was not discovered by our induction system.

Algorithm 3: Mapping of inventories.

type sense = {word : String ; `/* target word */`
cluster : set(String, float) } ; `/* set of word and weight pairs */`

input : $U \leftarrow$ a set of user senses
$T \leftarrow$ a set of TWSI senses
output: $mapping : U \mapsto T \cup \{\texttt{None}\}$ of user senses to TWSI senses

foreach $u \in U$ **do**
  $T_{u.word} \leftarrow \{t \in T \mid t.word = u.word\}$
  foreach $t \in T_{u.word}$ **do**
    $scores(t) \leftarrow cosine\_sim(t.cluster, u.cluster)$
  end
  if *at least one element of* $scores(t)$ *is* > 0 **then**
    $mapping(u) \leftarrow \underset{t \in T_{u.word}}{\operatorname{argmax}} scores(t)$
  end
  else
    $mapping(u) \leftarrow \texttt{None}$
  end
end

Metrics

Now that we have a mapping from user sense inventory to the TWSI inventory, we can evaluate predictions in a straightforward way. We use traditional *Precision*, *Recall* and *F-score* measures. Let $S = \{s_1, \ldots, s_n\}$, $G = \{g_1, \ldots, g_n\}$ and $P = \{p_1, \ldots, p_n\}$ be a set of test sentences, gold labels and

predicted labels respectively. For some sentences there might be no prediction: $|S| = |G| \geq |P|$. We define:

$$Precision = \frac{correct}{retrieved}; \qquad Recall = \frac{correct}{|S|}; \qquad F\text{-}score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall},$$

where

- *correct* is a number of (g_i, p_i) pairs for which $mapping(p_i) = g_i$
- *retrieved* $= |P|$ is a number of provided predictions

Note that if a sense was predicted by our system, but it has no mapping to TWSI senses, then this instance is counted as retrieved, but not correct.

Baselines

To facilitate the interpretation of the results we use several baselines:

- **Upper bound of the induced inventory**
  It always selects the correct sense for the context, but only if a mapping from the induced sense inventory to this sense exists. Therefore, a disambiguation system cannot perform better using a given sense inventory.
- **MFS of the TWSI inventory**
  This measure assigns the most frequent sense in the TWSI dataset. The most frequent sense is calculated on the full TWSI set. This baseline is usually hard to beat.
- **MFS of the induced inventory** assigns the identifier of the largest sense cluster of a target word. In a scenario like ours, when there is no information about sense frequency, the size of a sense cluster provides a good approximation of MFS.
- **Random sense baseline** of the TWSI and induced sense inventories selects a random sense for a target word according to the provided inventory.

6.3 Experiments

Our experiments are focused on the following stages of the system: construction of the similarity graph (Stage 2), clustering of ego-networks (Stage 3), pooling of word vectors (Stage 4) and word sense disambiguation. In particular, in Subsection 6.3.1 we investigate how the granularity of sense inventories affects the performance of disambiguation. In Subsection 6.3.2 we experiment with context filtering. In Subsection 6.3.3 we compare different similarity measures for computation of the word similarity graph, two methods for pooling of word vectors into sense vectors and two disambiguation strategies. We conclude this section with a test of statistical significance of our results and a conclusion, which summarizes our findings.

Parameters for Stage 1 remain unchanged for the whole set of these experiments. We have trained vectors on the complete English Wikipedia dump of October 2015. This corpus has a size suitable for the task (12 GB), provides broad coverage of word senses and has acquired popularity within the research community. We have preprocessed it with our custom script, meaning, among others, that both lower- and upper-cased spelling of words get a vector representation. We have

trained vectors using the following parameters of *word2vec*: CBOW model, window size 3, vector size 300, min-count 5, negative sampling 25, threshold for down-sampling of frequent words of 10^{-4} and 3 iterations. In the choice of the model, window size and vector size our decision was governed by a short series of preliminary experiments, which revealed that these settings lead to better performance of our system. For other parameters we used the default *word2vec* settings. The number of iterations is reasonable for the corpus of this size. We have saved both word and context vectors, each with a vocabulary of approximately 2.7 million words.

6.3.1 Sense Inventories

The goal of this experiment is to investigate how the granularity of sense inventories influences WSD performance. To achieve it, we take six different sense inventories, calculate sense vectors (Stage 4) and perform word sense disambiguation with fixed parameters on the sense-balanced TWSI dataset.

Setup

The first sense inventory was induced by our system using *word2vec* (w2v) similarities for calculation of nearest neighbours at Stage 2. We have applied filtering of words to speed up the process: during construction of the word similarity graph we have included words consisting only of lower- or upper-cased letters, a point or a dash, and before sense induction we have filtered out words with frequencies lower than 100. This mostly leads to removal of tokens resulted from tokenization errors, numbers, and infrequent named entities. For clustering of ego-networks we have used parameters $N = 200$, $n = 200$ and minimal cluster size $k = 5$.

The second one was produced by *JoBimText* framework using *JoBimText* (JBT) similarities for calculation of word neighbours. The word similarity model was built from English Wikipedia corpus using dependency features obtained with Malt parser. Same clustering parameters were used, except for minimal cluster size $k = 15$.

Another three inventories were also built with *JoBimText* using the same Wikipedia corpus and a different parser for extraction of dependency features (Stanford parser). For these three the ego-network connectivity parameter was varied ($n = 200$, $n = 100$ and $n = 50$), leading to increase in sense granularity. Finally, we also use the golden TWSI inventory, because it serves as a good baseline for word sense induction part of our task.

For calculation of sense vectors we applied an unweighted average of word vectors. We perform word sense disambiguation using probability strategy. The context is not filtered.

Results and Interpretation

Results of this experiment are presented in Table 6.2. The first two columns correspond to the inventory name and its average number of senses per word. The second two columns show the upper bound measure of an inventory and its WSD performance.

The upper bound reflects the best possible performance of the inventory based only on correctness of sense mapping, disregarding the correct or incorrect choice of a sense by the WSD mechanism. As it can be expected, the upper bound of the TWSI inventory equals 1 across all measures, because it provides exactly the same senses as used in the dataset and ensures an absolutely correct mapping of senses. Also unsurprisingly, it outperforms all other inventories on the WSD task. On the other hand, the F-score for the TWSI inventory in word sense disambiguation reaches only 0.484, which means that there is much room for improvement of our WSD mechanism.

Inventory	#Senses	Upper bound			Probability-based WSD		
		Prec.	Recall	F-score	Prec.	Recall	F-score
TWSI	2.26	1.000	1.000	1.000	0.484	0.483	0.484
w2v wiki, $n = 200$, $k = 5$	1.56	1.000	0.479	0.648	0.367	0.366	0.366
JBT wiki, Malt, $n = 200$, $k = 15$	1.64	1.000	0.488	0.656	**0.383**	**0.383**	**0.383**
JBT wiki, Stanford, $n = 200$, $k = 5$	2.55	1.000	0.598	0.748	0.338	0.338	0.338
JBT wiki, Stanford, $n = 100$, $k = 5$	3.59	1.000	0.671	0.803	0.305	0.305	0.305
JBT wiki, Stanford, $n = 50$, $k = 5$	5.13	**1.000**	**0.724**	**0.840**	0.275	0.275	0.275

Table 6.2: Influence of sense granularity on upper bound and WSD performance on sense-balanced TWSI dataset. Sense vectors calculated using unweighted pooling, disambiguation performed with probability strategy, no context filtering.

The other five inventories display sense granularity ranging from 1.56 to 5.13. Our first observation is that the more granular the sense inventory, the better match between the TWSI and the induced inventory can be established. It is quite natural, if we remember that we map every induced sense to at most one TWSI sense and therefore, having more senses per word increases our chances to cover more TWSI senses. If we consider that in the upper bound baseline precision always equals 1 (because prediction is always assumed to be correct), we can understand that recall in this scenario reflects the coverage of TWSI senses by an induced sense inventory.

However, the relation of actual WSD performance to granularity is inverse: the higher the number of senses, the lower the WSD performance, falling as low as 0.275 for the most granular *JBT wiki, Stanford, n=50, k=5* inventory with 5.13 senses per word on average. This is observed for all sense inventories except for *JBT wiki, Malt, n=200, k=15*, which despite having more senses on average than a comparable *w2v* inventory, delivers slightly better results in WSD. Based on this observation we speculate that having many senses per word (especially more senses per word than a golden TWSI inventory) makes sense disambiguation more complicated.

Also we see that in WSD tests our system produces almost identical precision and recall results. An explanation for this is the following: our WSD mechanism has no notion of prediction confidence. It always produces a prediction for a given test instance with respect to the induced sense inventory, even when this has no mapping to the TWSI inventory or the choice is not certain. As a result, precision always equals recall (except when the target word is not covered by our sense vectors and the prediction cannot be made at all). Introducing a measure of system's confidence of prediction and setting a threshold on it could increase the precision, but we have left it for future work.

In our further experiments we focus on first two sense inventories: one based on the w2v similarities and another based on JBT similarities (Malt parser). Because of almost identical training corpus, preprocessing and clustering parameters used to obtain these inventories, we consider them suitable for comparison of two measures of word similarity proposed for Stage 2 of our method.

	Full TWSI		Balanced TWSI	
	w2v	**JBT**	**w2v**	**JBT**
no filter	0.676	0.669	0.383	0.397
filter, $p = 5$	0.679	0.674	0.386	0.403
filter, $p = 3$	0.681	0.676	0.387	0.409
filter, $p = 2$	**0.683**	**0.678**	**0.389**	**0.410**
filter, $p = 1$	**0.683**	0.676	**0.390**	**0.410**

Table 6.3: Influence of context filtering on disambiguation in terms of F-score. Results for the sense-unbalanced (Full TWSI) and the sense-balanced (Balanced TWSI) dataset using w2v and JBT sense inventories. Sense vectors calculated using weighted pooling, disambiguation performed with similarity strategy.

6.3.2 Context Filtering

As we have observed from the gap in performance of the TWSI inventory on WSD task as compared to its upper bound, our WSD mechanism could be improved. One possibility to do so is through context filtering. The goal of this experiment is to find an optimal number of context words to use for disambiguation.

Setup

In this experiment we use two sense inventories obtained using *word2vec* similarities (w2v) and *JoBimText* similarities (JBT) and described in the previous subsection. We calculate sense vectors using weighted average of word vectors and perform disambiguation with similarity strategy. We repeat tests on both the sense-unbalanced and the sense-balanced TWSI datasets. We experiment with different number of words used for disambiguation: $p = 5$, $p = 3$, $p = 2$ and $p = 1$.

Results and Interpretation

Results of this experiment are presented in Table 6.3. We provide only the values of F-score measure because, as mentioned in the previous subsection, our system performs equally in terms of precision and recall.

First of all, we can observe that, as expected, context filtering improves the performance of WSD mechanism on both datasets independently of word similarity type. Secondly, we see the correlation between number of context words used for disambiguation and the results: with lower p value the F-score increases. This tendency stops around $p = 2$, which agrees with intuition: using only one, even most discriminating context word, might not be enough for correct prediction. Based on this experiment we suggest to use two words from context as an optimal setting for parameter p.

6.3.3 Various System Configurations

This is our largest experiment on the TWSI dataset. Its goal is to consistently compare all major variations of system's configuration and to summarize our findings. In particular, we investigate the

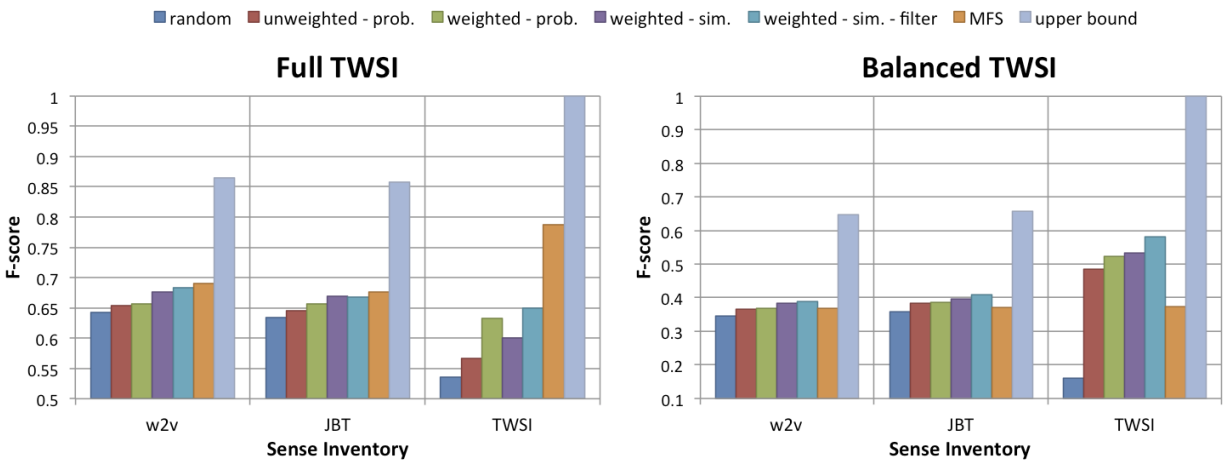


Figure 6.2: Visualisation of WSD performance (see Table 6.4) of different configuration of our method for three sense inventories (w2v, JBT and TWSI). Results for the sense-unbalanced TWSI dataset (Full TWSI) and the sense-balanced TWSI dataset (Balanced TWSI).

influence of two similarity measures, compare unweighted and weighted pooling of word vectors and contrast two disambiguation strategies. To facilitate the interpretation of results we include several baselines.

Setup

For this experiments we use three different inventories: two based on *word2vec* similarities (w2v) and *JoBimText* similarities (JBT), as described in Subsection 6.3.1, and the golden TWSI inventory. For each inventory we test four configurations of the system: (1) unweighted pooling of word vectors and probability disambiguation strategy, (2) weighted pooling of vectors and probability disambiguation strategy, (3) weighted pooling of vectors and similarity disambiguation strategy and (4) weighted pooling of vectors and similarity disambiguation strategy with optimal context filtering ($p = 2$). Also for each inventory we compute three baselines: (1) upper bound of the inventory, (2) most frequent sense baseline and (3) random sense baseline. All test are repeated on the sense-unbalanced (full TWSI) and the sense-balanced TWSI datasets.

Results and Interpretation

Results of this experiment are presented in Table 6.4. First three blocks of lines correspond to three baselines. The last three blocks correspond to three inventories, with four model configurations in each. Results in terms of F-score are visualised in Figure 6.2.

First of all, we look at the baselines. We may observe that our system significantly outperforms the random sense baseline of all inventories on both datasets. This is especially noticeable for the TWSI inventory which has the largest number of senses per word. With regard to the most frequent sense baseline, which is usually very difficult to outperform, our models based on induced inventories are performing comparably, only slightly loosing on the sense-unbalanced dataset.

Then we focus on the different pooling and disambiguation methods. First of all, we see that using weighted pooling of word vectors in a sense cluster (line *weighted – prob.*) consistently produces better results across all inventories and datasets in comparison to the unweighted pooling

(line *unweighted – prob.*). Indeed, some clusters may contain irrelevant words and using weights helps to discount their contribution.

Moreover, the similarity disambiguation strategy (line *weighted – sim.*) always yields better results than probability disambiguation strategy (line *weighted –prob.*). This complies with an observation that cosine similarity between word vectors proved to be useful for semantic relatedness [Mikolov et al., 2013a, Baroni et al., 2014]. At the same time, there is less evidence about successful use-cases of the CBOW language model itself, on which the probability strategy is based.

Finally, this experiment confirms that filtering of context words (line *weighted – sim. – filter ($p = 2$)*) improves the performance of our disambiguation mechanism. This is again observed across all inventories and datasets.

Last but not least, we look at the relevance of word similarity measure used for induction of sense (w2v inventory vs. JBT inventory). They perform comparably, with JBT based inventory producing slightly better results on the balanced dataset (0.41 vs. 0.39). Quite surprisingly, using the gold TWSI inventory improves the results in comparison to the induced inventories only on the sense-balanced dataset. It is not obvious, why it looses to the induced inventories on the unbalanced full TWSI dataset.

6.3.4 Statistical Significance of the Results

Our experiments provide indications about optimal settings for various parameters of our model. However, it is noticeable that in many cases, the the difference in result values is very small. For example, the context filtering improves the performance of the model based on the w2v inventory from 0.676 to 0.683 (F-score on Full TWSI). In order to understand if this small change is sufficient to state that one configuration performs better the other, we can apply a statistical significance test. Such test can indicate whether the null hypothesis about two sets of values holds.

Theory

The *null hypothesis* refers to a statement that the change in values of a binary variable at two points in time over the same population cannot be explained by anything rather than chance. Therefore our goal is to refute the null hypothesis about performance of our models. In our scenario the *population* is a collection of sentences (test instances) in a dataset. Two models we want to compare can be considered as two *points in time*. In this case a *binary variable* is a `True`/`False` value assigned to a prediction produced by a model for a sentence in the dataset (`True` for a correct prediction and `False` for a wrong prediction).

There exist various statistic tests designed to check if the null hypothesis holds. One of them is McNemar test introduced in 1947 [McNemar, 1947]. It is applied to a $2x2$ contingency table with the outcomes of two model tests. Omitting the table itself, we define two relevant values:

- b – the number of sentences correctly predicted by the first, but not the second model.
- c – the number of sentences correctly predicted by the second, but not the first model.

There are several formulations of the McNemar test statistic. The original one is given as follows:

$$\chi^2 = \frac{(b-c)^2}{b+c}.$$

Under the null hypothesis the function χ^2 approximates the chi-squared distribution with one degree of freedom. However, if χ^2 is improbably large according to chi-squared distribution, then

Model	#Senses	Full TWSI			Balanced TWSI		
		Prec.	Recall	F-score	Prec.	Recall	F-score
Upper bound of the TWSI inventory	2.26	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
Upper bound of the w2v inventory	1.56	1.000	0.761	0.864	1.000	0.477	0.646
Upper bound of the JBT inventory	1.64	1.000	0.751	0.858	1.000	0.488	0.656
MFS of the TWSI inventory	2.26	**0.787**	**0.787**	**0.787**	**0.373**	**0.373**	**0.373**
MFS of the w2v inventory	1.56	0.691	0.690	0.690	0.370	0.369	0.369
MFS of the JBT inventory	1.64	0.677	0.676	0.676	0.370	0.370	0.370
Random Sense of the TWSI inventory	2.26	0.536	0.534	0.535	0.160	0.160	0.160
Random Sense of the w2v inventory	1.56	**0.643**	**0.643**	**0.643**	0.347	0.346	0.346
Random Sense of the JBT inventory	1.64	0.634	0.634	0.634	**0.359**	**0.359**	**0.359**
w2v – unweighted – prob.	1.56	0.655	0.654	0.654	0.367	0.366	0.366
w2v – weighted – prob.	1.56	0.657	0.656	0.656	0.367	0.367	0.367
w2v – weighted – sim.	1.56	0.677	0.676	0.676	0.383	0.382	0.383
w2v – weighted – sim. – filter ($p = 2$)	1.56	**0.683**	**0.682**	**0.683**	**0.390**	**0.389**	**0.389**
JBT – unweighted – prob.	1.64	0.646	0.646	0.646	0.383	0.383	0.383
JBT – weighted – prob.	1.64	0.657	0.657	0.657	0.387	0.387	0.387
JBT – weighted – sim.	1.64	0.669	0.669	0.669	0.397	0.397	0.397
JBT – weighted – sim. – filter ($p = 2$)	1.64	**0.678**	**0.678**	**0.678**	**0.410**	**0.410**	**0.410**
TWSI – unweighted – prob.	2.26	0.568	0.565	0.567	0.484	0.483	0.484
TWSI – weighted – prob.	2.26	0.634	0.631	0.633	0.522	0.521	0.522
TWSI – weighted – sim.	2.26	0.602	0.599	0.601	0.533	0.532	0.532
TWSI – weighted – sim. – filter ($p = 2$)	2.26	**0.650**	**0.648**	**0.649**	**0.582**	**0.582**	**0.582**

Table 6.4: WSD performance of different configuration of our method for three sense inventories (w2v, JBT and TWSI). Results for the sense-unbalanced TWSI dataset (Full TWSI) and the sense-balanced TWSI dataset (Balanced TWSI). Model name denotes its configuration: inventory – pooling method – disambiguation strategy – context filtering. The best results per section are typeset in boldface.

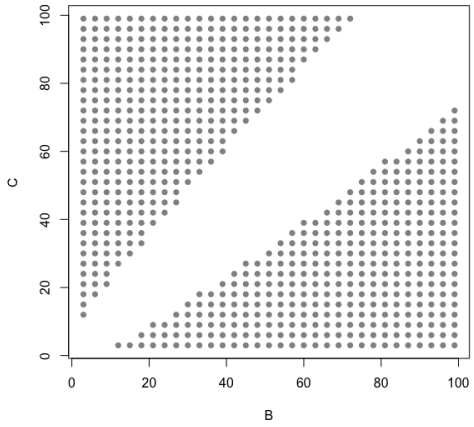


Figure 6.3: p_{mid} measure over $b, c \in [1, 100]$. Grey area corresponds to $p_{mid} < 0.05$.

the null hypothesis may be rejected. In other words, a larger value χ^2 means a greater significance of the result.

For small values of either b or c ($b + c < 25$), χ^2 does not provide a good approximation of imbalance between them. In this case one of the following *p-measures* can be used. They make an assumption that the data under null hypothesis can be explained by a binomial distribution. Therefore, if the probability of actual observations under this assumption is too low, then the assumption may be considered wrong.

$$p_{exact} = \texttt{binom.cdf}(b),$$
$$p_{mid} = p_{exact} - \texttt{binom.pmf}(b).$$

Here `binom` is initialized with shape parameters $n = b + c$ and $p = 0.5$. `cdf` is a cumulative distribution function and `pmf` is the probability mass function. Lower values of p indicate higher statistical significance. Usually the null hypothesis is rejected when $p < 0.05$. Figure 6.3 illustrates the behaviour of p_{mid} measure for b and c in the range $[0, 100]$. Any (b, c) pair falling in to the grey area is considered statistically significant.

Results

We have applied this test on six pairs of models, testing the importance of context filtering and sense inventory types. We have decided to use the p_{mid} measure because evidence exists that it works better than others [Fagerland et al., 2013]. Results of our tests are given in Table 6.5. Each row corresponds to a pair of models which performance variation is being investigated. In the first column are the parameters which were the same for both tested model. The second column tells what parameter was varied. We report the p_{mid} value, calculated for model predictions over development and full TWSI datasets.

All results in the table are lower than 0.05, from which we conclude that null hypothesis can be refuted and differences in experimental results are statistically significant.

Model		P-measure (p_{mid})	
Fixed parameters	Altered parameter	Full TWSI	Dev TWSI
w2v – weighted – sim	no filter / filter ($p = 2$)	<0.001	0.013
JBT – weighted – sim	no filter / filter ($p = 2$)	<0.001	<0.001
TWSI – weighted – sim	no filter / filter ($p = 2$)	<0.001	<0.001
weighted – sim – filter ($p = 2$)	w2v / JBT	0.001	0.002
weighted – sim – filter ($p = 2$)	w2v / TWSI	<0.001	<0.001
weighted – sim – filter ($p = 2$)	TWSI / JBT	<0.001	<0.001

Table 6.5: Statistical significance of differences in performance of four pairs of models expressed as p_{mid} measure. Any value below 0.05 indicates that the difference is statistically significant.

6.3.5 Conclusion

The series of experiments on TWSI has allowed us to find optimal setting for some of our parameters. We suggest to use the weighted pooling method at Stage 4 of our method and the similarity-based disambiguation strategy with context filtering. With regard to the word similarity measure (*word2vec* similarity or *JBT* similarity), no clear choice can be made, as one measure outperforms the other depending on the dataset. To support our conclusion we have performed the significance test over the experimental results. While we run this test only with regard to the context filtering and inventories, we assume that it would similarly hold for variations between pooling methods and disambiguation strategies.

6.4 Error Analysis

In Chapters 4 and 5, which were dedicated to our word sense induction and word sense disambiguation mechanisms, we have provided several examples of the successful performance of our system. However, as evident from the experiments presented in this chapter, it is not always able to accomplish its task. To find out the reasons for it, we perform manual error analysis. With regard to the design of our system, we have found it appropriate to investigate the output of several system's stages. In particular, we take a look at the word similarity graph (in other words, 200 neighbours of polysemous words), sense clusters and instances of the dataset, that where assigned a wrong sense label.

Problems with Word Neighbours

In many cases our system makes a prediction error, because the correct sense has not been induced and is not in the inventory on which the system operates. The induction capacity of our system depends on the quality of extracted neighbours and the clustering performance. First we observe the neighbours.

One of our expectations, on which the induction process is based, is that 200 nearest neighbours of a word contain terms related to the word's different senses. By looking at the distributional

		Neighbours	
Word	**Expected Senses**	**w2v**	**JBT**
Ruby	name, language	Daisy, Violet, Lily, Cora, Jasmine, Amber, Maggie, Molly, Dee, Holly.	Jade, Ellie, Chloe, Ada, Debbie, Perl, Python, PHP, Java, Haskell.
architecture	art, structure	Neo-Classical, neoclassical, Indo-Saracenic, classicism, Neo-Byzantine, architectonic.	building, art, structure, literature, decor, processor, hardware.
bass	music, fish	drums, trumpet, guitar, keyboards, bodhran, autoharp, darbuka, cello.	guitar, piano, drum, violin, banjo, catfish, trout, carp, pike, perch.
club	sport, nightclub	team, clubs, league, Juventus, Hibs, FC, striker, Benfica, Dinamo, Bundesliga.	Arsenal, team, league, Millwall, venue, hall, nightclub, clubhouse.
magazine	journal, gun	magazines, periodical, e-zine, webzine, InStyle, newspaper, Newsweek.	publication, journal, book, fiction, Newspaper, blog, paper, newsletter.

Table 6.6: Excerpts of 200 nearest neighbours that do not contain terms related to multiple senses of a word. Examples provided for neighbours extracted with *word2vec* and *JoBimText* similarities. Neighbours are not ordered by weights.

thesauri build at Stage 2 with *word2vec* and *JoBimText* similarity measures, we found out that it is not always the case.

We have investigated the thesauri built from the Wikipedia corpus as described in the TWSI experiment. Some examples that illustrate our findings are presented in Table 6.6. For five words we have randomly picked several neighbours that approximate well the content of their 200 neighbours. Our choice of five words was governed by the problem most prominently evident in the neighbours extracted with *word2vec*.

If we look closely at neighbours extracted with *word2vec*, we can see that they contain terms related to only one of the expected word senses. For example, for *Ruby*, which may refer to a women name or the programming language, only names are included. The same applies to all five words in the table. Moreover, among extracted neighbours are many rare word, like *bodhran*, *darbuka* for *bass* and *e-zine*, *webzine* for *magazine*. From this we draw a conclusion, that the placement of words in the vector space by *word2vec* is highly influenced by their frequency and rare similar words are pulled together. When we train *word2vec* on a large corpus (12 GB compared to 1.5 GB used in the example in Section 4.6), the variety of rare terms, for example women names or musical instrument types, increases, and they take all the places in 200 extracted neighbours, leaving terms related to other senses out. Indeed, some names of programming languages could be observed among neighbours of *Ruby*, but at much lower ranks.

On the other hand, this problem is less noticeable for neighbours extracted with *JoBimText*. For the first four words in our table, *JoBimText* neighbours included terms related to both expected senses. Also they contain more general and more frequent terms, for example *art*, *building* for the word *architecture*, whereas *word2vec* extracts specific architecture styles. However, the discussed problem also occurs with *JoBimText* neighbours (see neighbours of *magazine*).

Word	Expected Senses	Neighbours
ruby	gem_1, $colour_2$	$sapphire_1$, $topaz_1$, $gemstone_1$, $amethyst_1$, $blood\text{-}red_2$, $sapphires_1$, $jewels_1$ $rubies_1$, $draconium_1$, $deep\text{-}blue_2$, $earring_1$, $rose\text{-}colored_2$, $purple_2$, $gemstones_1$.
instrument	$music_1$, $device_2$	$device_2$, $lute_1$, $method_2$, $lute\text{-}like_1$, $tool_2$, $organ_1$, $lyre_1$, $ukuleles_1$, $technique_2$, $thermometer_2$, $transducer_2$, $keyboard_1$, $apparatus_2$, $resonator_1$, $xylophones_1$.
chair	$person_1$, $furniture_2$	$chairman_1$, $chairperson_1$, $head_1$, $president_1$, $chairwoman_1$, $professorship_1$, $co\text{-}chair_1$, $vice\text{-}chair_1$, $dean_1$, $desk_2$, $sofa_2$, $under\text{-}secretary_1$, $chairmanship_1$.

Table 6.7: Excerpts of 200 nearest neighbours that contain terms related to multiple senses of a word, but were assigned to the same sense cluster. Examples provided for neighbours extracted with *word2vec*. Neighbours are not ordered by weights and are manually annotated with expected senses.

Problems with Sense Clusters

As stated above, the induction capacity of our system also depends on the clustering performance. To investigate errors at this stage, we have observed clusters produced by the Chinese Whispers algorithm from the word similarity graph based on *word2vec* similarity measure. We specifically looked for words whose 200 nearest neighbours are not subject to the problem discussed earlier, meaning that they contain term related to different senses.

We have found out that for some of these words the clustering does not split the ego-network in parts, even though representatives of different senses are present. Examples of such words are given in Table 6.7. Again, we show only several randomly picked neighbours of these words trying to preserve the ratio of sense-related neighbours. Neighbours in the table are hand-annotated with expected senses for readability. All 200 neighbours of these words were assigned to one cluster.

As we can see, a lower-cased word *ruby* may refer to a gemstone or colour, which is also confirmed by its neighbours, however, the clustering algorithm fails to recognise two separate sense clusters. The same can be observed for other words. In case of the word *chair*, one may notice that neighbours of one sense (*person*) heavily outnumber the neighbours of the other sense (*furniture*). Although clusters induced from *JoBimText*-based word similarity graph for the same words are not present in this table, we have found out that they, on the contrary, correspond to expected senses. However, this observation may not be suitable for fair comparison of *word2vec* and *JoBimText* similarity measures in terms of clustering performance, because we only looked up those words in JBT sense inventory, that were already found to be clustered incorrectly in w2v sense inventory.

Problems with Disambiguation

All problems discussed previously were related to the induction of senses. However, even when our system operates on the golden sense inventory, it only reaches 58% of precision on the sense-balanced TWSI dataset with the best configuration. Our goal is to understand the reasons behind this number. For this we focus on the errors that occur only at the disambiguation stage and cannot be explained by the incorrect induction of senses.

First, we try to understand if the TWSI inventory, while providing the correct distinction of senses, might be inferior to w2v inventory in terms of representation of senses. Our WSD mechanism uses representation of senses in the vector space. The *word2vec*-based inventory provides sense clusters derived directly from the vector space on which the disambiguation operates. The

Word	Senses	Precision			
		Full TWSI		Balanced TWSI	
		w2v	TWSI	w2v	TWSI
metal	music, material	0.94	0.89	1.0	1.0
solution	mixture, problem	0.91	0.88	1.0	0.9
plant	factory, biology	0.86	0.92	0.9	0.9
performance	effectiveness, show	0.85	0.84	0.8	0.8
head	chair, body	0.77	0.87	0.9	1.0
	Average:	0.86	0.88	0.92	0.92

Table 6.8: Correctly induced words in the w2v inventory and disambiguation precision on these words by models based on the w2v and TWSI inventories. Configuration of models is *weighted – sim. – filter (p = 2)*.

TWSI sense clusters, on the other hand, are built on a different notion of word substitutions. To understand if this is a relevant difference, we have made the following experiment.

We hand-pick several words, which senses are correctly induced by our system from a *word2vec*-based word similarity graph. We define correct as:

- TWSI and w2v inventories have an equal number of senses for a given word,
- the mapping function from w2v senses to TWSI senses of a given word is bijective (all senses of TWSI inventory are mapped to).

Afterwards we calculate precision of the *word2vec*-based model and the TWSI-based models for those words only. We use our best configuration: *weighted – sim. – filter (p = 2)* run on the full and balanced TWSI datasets.

Results of this experiment are presented in Table 6.8. Both models perform almost equally, from which we conclude that First, we can conclude that, when induction errors are taken out of the picture, the choice of inventory does not influence the performance of disambiguation mechanism. Second, there is no evidence that disambiguation errors of the TWSI-based model are caused by the nature of the TWSI inventory (word substitutions).

Therefore, we draw our attention to the context of instances, where the TWSI model cannot identify the correct sense. We have looked at multiple negative examples and have noticed two patterns. First of all, our model can be mislead by some words in the context. For example, in a sentence "*Most trans fats consumed today are industrially created by partially hydrogenating plant oils – a process developed in the early 1900's and first commercialized as Crisco in 1911.*" the word *plant* is used in the *biology* meaning, however the model predicts the *factory* sense because of the context words *industrially* and *hydrogenating*.

Secondly, many of the disambiguation errors are concentrated at the words that have very close senses, often represented with overlapping sense clusters. For example the word *judge* has two senses: one related to *sports* and the other related to the *court of law*. However, their sense cluster share terms like *juror* and *arbiter*. As the result, the corresponding sense vectors are very similar and cannot effectively disambiguate words even in contexts that provide enough clues.

7 Evaluation on SemEval

In the previous chapter we have evaluated our system on the less frequently used, but large TWSI dataset. We have compared various configurations of the system and were able to identify optimal values for some of the parameters. However, not less important is to understand how well our system performs against other state-of-the-art approaches. To achieve this we have decided to test it on the word sense induction and disambiguation task of the SemEval 2013 competition (Task 13). The results of this experiment are described in this chapter.

This chapter is structured as follows: In the first section we introduce the dataset, evaluation metrics and other participants of the competition. In the second section we present results of the experiment.

7.1 Dataset and Evaluation Metrics

7.1.1 Dataset

SemEval 2013 Task 13 [Jurgens and Klapaftis, 2013] was part of an on-going series of Semantic Evaluation (SemEval) competitions dedicated to the evaluation of semantic analysis systems. It has has been designed for a setting, where each target word in a sentence may be annotated with several sense labels weighted by the likelihood of their applicability in the given context. The motivation for this idea came from an observation that annotators tend to find several applicable senses for one word if a fine-grained sense inventory, such as WordNet, is used [Erk and McCarthy, 2009, Passonneau et al., 2012, Jurgens, 2013]. Nevertheless, the dataset can be used for single-sense annotation too.

The dataset provides the total of 4664 sentences for 20 nouns, 20 verbs and 10 adjectives. Target words are manually annotated with WordNet senses, with an average of 1.12 labels per instance. The minimum of 22 sentences and the maximum of 100 sentences per word are given. All sentences were drawn from Open American National Corpus [Ide et al., 2004], including spoken and written portions of the corpus.

The participants were asked to annotate target words with sets of either WordNet sense labels or labels of an induced sense inventory. Labels should have applicability weights: measure of how suitable a sense is in the given context. The task was designed for unsupervised systems, therefore no training data was provided. For sense induction participants were allowed to use the *ukWaC* corpus [Baroni et al., 2009]: a 2 billion word English corpus crawled from the .uk domain of the web.

7.1.2 Evaluation Metrics

As it has been mentioned previously, a great challenge in evaluation of WSD systems operating on an induced sense inventory is the mapping of gold and induced inventories. For the SemEval 2013 Task 13 five evaluation metrics were proposed based on two evaluation approaches: supervised and unsupervised. In this subsection we explain the approaches and present the metrics.

Supervised Evaluation

Supervised evaluation approaches are based on the mapping of two inventories. Even though similar in name, this approach is different from the direct mapping of two inventories based on sense cluster overlap that we have used for TWSI evaluation. Instead, the mapping is learned from the dataset already labelled by the participating system. The dataset is randomly divided into five partitions, four of which are used to learn the mapping of golden and induced senses, and the last part is used for evaluation. This process is repeated five times, so that each partition could be evaluated once. This approach makes two evaluation metrics applicable.

The *Jaccard Index* measures the agreement of chosen labels. It equals $\frac{|X \cap Y|}{|X \cup Y|}$, where X is a set of assigned sense lables and Y is a set of golden sense labels. This measure disregards weights of sense labels.

The *Positionally weighted Kendall's* τ evaluates the ranking of senses assigned to the instance. It employs a modification of Kendall's τ: a distance measure that counts the number of swaps necessary to make two sequence identical, with positional weighting that penalises errors in higher ranks more than errors in lower ranks [Kumar and Vassilvitskii, 2010]. The measure is normalized by the maximal possible distance, which depends on the sequence length, and transformed to similarity measure.

The goal of *Weighted Normalised DCG (WNDCG)* is to evaluate the agreement in applicability weights assigned to sense labels. The measure is adapted from Information Retrieval setting. It is a weighted version of Normalised Discounted Cumulative Gain (NDCG) [Moffat and Zobel, 2008] that takes into account the weights assigned to the predicted labels.

Unsupervised Evaluation

Unsupervised approaches are based on the idea that the WSD task can be regarded as a clustering challenge. Each set of instances for one target word has to be clustered into groups representing senses of this word. Therefore, it can be evaluated with measures developed for clustering evaluation. The special aspects of the tasks itself are that: (1) clusters can overlap, because one instance may be labelled with several senses and (2) clusters are fuzzy, because applicability of a sense for a given instance is expressed with a weight.

Fuzzy B-cubed is a measure that compares two clusterings on a per-item basis, integrating the notion of precision and recall. It is based on the B-cubed measure version that can handle overlapping clusters [Amigó et al., 2009] and was further adapted to work with fuzzy clusters, where degree of cluster membership is specified by a weight.

Fuzzy Normalised Mutual Information (Fuzzy NMI) is an extension of Mutual information, the quantity used to measure the level of agreement between two variables, which in this scenario represent gold and assigned sense labels. It has been adapted from [Lancichinetti et al., 2009] version to work on fuzzy overlapping clusters. It evaluates performance on the clusters level rather than the instances.

All of the described here measures are normalised to $[0, 1]$. Their formal definitions, as well as accompanying detailed explanations, can be found in [Jurgens and Klapaftis, 2013].

7.1.3 Participants

Four participants have submitted nine models in total to the competition. *AI-KU* [Baskaya et al., 2013] is a system which induces senses based on lexical substitutions of target words. It first collects 100 substitutions for a target words in each instance. Afterwards, all instances, represented with a

vector based on these substitutions, are clustered by the k-means algorithm. By design, this system is not able to assign labels to new single instances.

Unimelb [Lau et al., 2013] applies Hierarchical Dirichlet Process to discover topics of target words from a corpus. For this they use contextual and positional features. Discovered topics are interpreted as word senses. To perform disambiguation, they calculate the probability of a topic applicability in a given context.

UoS [Hope and Keller, 2013b] is based on dependency-features of words. For each target word in a SemEval dataset they build a graph of 300 hundred words appearing in a dependency relation with respective target word. Afterwards, they use MaxMax algorithm to produce sense clusters from these graphs. For disambiguation they calculate a separation score between context words and the words of the sense cluster.

La Sapienza [Agirre and Soroa, 2009] is the only system that does not perform word sense induction and instead relies on the WordNet sense inventory. It applies Personalized Page Rank algorithm on a WordNet-based network to compare the similarity of each sense with the similarity of the context.

7.2 Experiment

In this experiment we evaluate the best configurations of our system on the SemEval dataset. We variate the type of similarity measure used to calculate the word similarity graph (w2v or jbt), but keep all other parameters fixed. As in previous experiments, models based on the TWSI dataset are also included.

Setup

In order to enable clean comparison of our system to the participants of SemEval competition, we retrain the models on the ukWaC corpus. The parameters of *word2vec* used at the first stage remain unchanged (CBOW model, context window 3, min-count 5, 3 iterations), except for the size of vectors, which equals 100. We again use w2v- and JBT- based approaches for induction of senses. All procedures, including parameters of the Chinese Whispers clustering algorithm, remain the same as for TWSI experiments (see details in Subsection 6.3.1, Setup paragraph). Based on the findings of the previous experiment, we calculate sense vectors with weighted pooling method and apply similarity disambiguation strategy with filtering of context words ($p = 2$).

We test our models on the full SemEval dataset. Because the TWSI inventory contains only nouns, we can apply it only to the subset of SemEval dataset, which consists of instances where the target word is a noun. For comparison, we run our w2v- and JBT-based configurations on such subset as well.

We compare our system to the *AdaGram* system, which was described in Related Work Section 2.1.2. As it has not participated in the SemEval competition, we have trained it ourselves on the ukWaC corpus. The results delivered by SemEval participants were taken from the original paper reporting on this competition [Jurgens and Klapaftis, 2013].

Results and Interpretation

Results of this comparative evaluation are presented in Table 7.1. First of all, one may observe that dependency-based (JBT) word similarities yield slightly better results than word embedding similarity (w2v) for inventory induction. Secondly, we see that our model based on the TWSI

Model		Supervised			Unsupervised	
		Jacc.	Tau	WNDCG	F.NMI	F.B-Cubed
Baselines	One sense for all	0.171	0.627	0.302	0.000	0.631
	One sense per instance	0.000	0.953	0.000	0.072	0.000
	Most Frequent Sense (MFS)	0.579	0.583	0.431	–	–
SemEval	AI-KU (add1000)	0.176	0.609	0.205	0.033	0.317
	AI-KU	0.176	0.619	0.393	0.066	0.382
	AI-KU (remove5-add1000)	0.228	0.654	0.330	0.040	0.463
	Unimelb (5p)	0.198	0.623	0.374	0.056	0.475
	Unimelb (50k)	0.198	0.633	0.384	0.060	0.494
	UoS (#WN senses)	0.171	0.600	0.298	0.046	0.186
	UoS (top-3)	0.220	0.637	0.370	0.044	0.451
	La Sapienza (1)	0.131	0.544	0.332	–	–
	La Sapienza (2)	0.131	0.535	0.394	–	–
Sense emb.	AdaGram, $\alpha = 0.05$, 100 dim. vectors	0.274	0.644	0.318	0.058	0.470
Our models	w2v – weighted – sim. – filter ($p = 2$)	0.197	0.615	0.291	0.011	0.615
	w2v – weighted – sim. – filter ($p = 2$): nouns	0.179	0.626	0.304	0.011	0.623
	JBT – weighted – sim. – filter ($p = 2$)	0.205	0.624	0.291	0.017	0.598
	JBT – weighted – sim. – filter ($p = 2$): nouns	0.198	0.643	0.310	0.031	0.595
	TWSI – weighted – sim. – filter ($p = 2$): nouns	0.215	0.651	0.318	0.030	0.573

Table 7.1: Performance of the best configurations of our method on the SemEval 2013 Task 13 dataset. Model name denotes its configuration: inventory type – pooling method – disambiguation strategy – context filtering.

inventory significantly outperforms both JBT- and w2v-based models. This indicates that precise sense inventories greatly influence the quality of WSD performance.

In general, performance of the best configurations of our method is comparable to the top-ranked SemEval participants, but is not systematically exceeding their results. AdaGram sometimes outperforms our method, sometimes it is on par, depending on the metric. We interpret these results as an indication of comparability of our method to state-of-the-art approaches.

Finally, note that none of the unsupervised WSD methods discussed in this paper, including the top-ranked SemEval submissions and AdaGram, were able to beat the most frequent sense baselines of the respective datasets (with the exception of the balanced version of TWSI). The similar results are observed for other unsupervised word sense disambiguation methods [Nieto Piña and Johansson, 2016].

8 Conclusion and Future Work

8.1 Conclusion

In this work we have built an unsupervised knowledge-free words sense induction and disambiguation system. In this we have achieved our main goal.

Senses in our system are induced for each word in separate via clustering of its ego-network. Different measures of similarity can be used to construct the word ego-network, of which we have tried *JoBimText* word similarity and cosine vector similarity of word embeddings obtained with *word2vec* toolkit. In general, the mechanism heavily relies on the distributional theory. Each sense may be represented as a cluster of related terms or a sense embedding. The number of senses is induced automatically by the clustering algorithm. For induction of senses only a large text corpus is needed.

The disambiguation mechanisms of our system works on vector representations of word senses and the context. The context is regarded as a bag of words. We have proposed two metrics for estimation of sense applicability in the given context: similarity-based and probability-based. In contrast to other state-of-the-art systems that disambiguate senses via clustering of test instances, our WSD approach can be used in the online setting.

We have performed two stages of evaluation. In the first one based on the TWSI dataset, we investigated the influence of different parameters. We have established (1) that sense vectors should be computed with weights of terms in the sense cluster accounted for, (2) that the similarity-based disambiguation mechanism yields better results than the probability-based one and (3) that context filtering based on words' relevance for the disambiguation improves the performance. Use of different word similarity measures at the induction step of our system does not show strong influence on the results, but integration of the correct sense inventory dramatically improves performance on the test dataset.

In the second stage we have evaluated our system on the SemEval 2013 Task 13 to enable comparison of our approach to the state-of-the-art systems. We were not able to outperform participants of this competition, but our results were competitive.

One significant advantage of our system is its flexibility. Most of its elements can be substituted by their better counterparts, should these emerge. This holds for word embedding training algorithms and clustering algorithms. In addition to this, one can directly use pre-trained word vectors for sense induction or representation of words during disambiguation, and available sense inventories for computation of sense vectors.

The most notable limitations of our system are the following. It is better suited for scenarios where coarse-grained distinction of senses is expected. In terms of scalability, its bottleneck is at the construction of the word-similarity graph based on *word2vec* vector similarity. It requires V^2 calculation of vector dot products, where V is the size of the vocabulary.

8.2 Future Work

In Section 6.4 we have discussed several flaws of our system. With respect to these we have suggestions for future work that could improve the performance of our system.

First of all, we have identified that our approach for construction of words' ego-networks is not always able to include neighbours of words related to their different senses. This problem is especially prominent, when *word2vec* word similarity measure is used. In order to overcome this problem, one could re-weight the similarity scores between word pairs before extraction of 200 nearest neighbours with some approximation of words frequency or information content. The goal is to lower the ranks of very specific infrequent neighbours of a word, that *word2vec* tends to pull very close.

Secondly, we have found out that even when an ego-network contains terms related to different senses, the Chinese Whispers clustering algorithm is sometimes not able to separate them into different clusters. For one, we have not experimented with the parameter of CW that controls granularity of clusters. Also, one could try other clustering algorithms like MCL [van Dongen, 2000] or MaxMax [Hope and Keller, 2013a].

Finally, we have discussed that even when provided with a golden sense inventory, our WSD mechanism is not able to reach 60% precision on the sense-balanced TWSI dataset. In order to improve the quality of sense disambiguation we suggest to use syntactic dependencies as context features. Thanks to the recently proposed modification of *word2vec* approach to training word embeddings developed by [Levy and Goldberg, 2014a], it is possible to adapt our WSD mechanism to this setting. Their algorithm allows to train word embeddings on arbitrary word/context pairs, including pairs of words and dependency relations they occurs in along the corpus. This provides us with with vectors that represent different dependency context features. The solution is then to parse a test sentence, extract dependencies in which the target word participates, represent those dependencies with available vectors and proceed in the same manner that our disambiguation mechanism identifies the correct word sense. Because dependency context features are known to be sparse, it is advisable to combine them with bag of words features. For example, this can be achieved by concatenation of two context vectors: one built from BoW features and another from dependency features.

Bibliography

[Agirre and Edmonds, 2007] Agirre, E. and Edmonds, P. (2007). *Word Sense Disambiguation: Algorithms and Applications*. Springer, Berlin, first edition.

[Agirre and Soroa, 2009] Agirre, E. and Soroa, A. (2009). Personalizing PageRank for word sense disambiguation. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 33–41, Athens, Greece.

[Amigó et al., 2009] Amigó, E., Gonzalo, J., Artiles, J., and Verdejo, F. (2009). A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12(4):461–486.

[Bansal et al., 2014] Bansal, M., Gimpel, K., and Livescu, K. (2014). Tailoring continuous word representations for dependency parsing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, Volume 2: Short Papers*, pages 809–815, Baltimore, MD, USA.

[Baroni et al., 2009] Baroni, M., Bernardini, S., Ferraresi, A., and Zanchetta, E. (2009). The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226.

[Baroni et al., 2014] Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014*, pages 238–247, Baltimore, MD, USA.

[Bartunov et al., 2015] Bartunov, S., Kondrashkin, D., Osokin, A., and Vetrov, D. (2015). Breaking sticks and ambiguities with adaptive Skip-gram. *arXiv preprint arXiv:1502.07257*.

[Baskaya et al., 2013] Baskaya, O., Sert, E., Cirik, V., and Yuret, D. (2013). AI-KU: Using substitute vectors and co-occurrence modeling for word sense induction and disambiguation. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 300–306, Atlanta, GA, USA.

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, 3:1137–1155.

[Biemann et al., 2013] Biemann, C., Riedl, and M. (2013). Text: Now in 2D! A framework for lexical expansion with contextual similarity. *Journal of Language Modelling*, 1(1):55–95.

[Biemann,] Biemann, C. Chinese Whispers: An efficient graph clustering algorithm and its application to natural language processing problems. In *Proceedings of the First Workshop on Graph Based Methods for Natural Language Processing*, pages 73–80, New York City, NY, USA.

[Biemann, 2012] Biemann, C. (2012). Turk bootstrap word sense inventory 2.0: A large-scale resource for lexical substitution. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC 2012)*, pages 4038–4042, Istanbul, Turkey.

[Brown, 2008] Brown, S. W. (2008). Choosing sense distinctions for WSD: Psycholinguistic evidence. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 249–252, Columbus, Ohio.

[Chen et al., 2014] Chen, X., Liu, Z., and Sun, M. (2014). A unified model for word sense representation and disambiguation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, pages 1025–1035, Doha, Qatar.

[Church and Hanks, 1990] Church, K. W. and Hanks, P. (1990). Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29.

[Collobert and Weston, 2008] Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning (ICML 2008)*, pages 160–167, Helsinki, Finland.

[Daelemans et al., 1999] Daelemans, W., Van Den Bosch, A., and Zavrel, J. (1999). Forgetting exceptions is harmful in language learning. *Machine Learning*, 34(1):11–41.

[Deerwester et al., 1990] Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391.

[Erk and McCarthy, 2009] Erk, K. and McCarthy, D. (2009). Graded word sense assignment. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1*, EMNLP '09, pages 440–449, Singapore.

[Fagerland et al., 2013] Fagerland, M. W., Lydersen, S., and Laake, P. (2013). The McNemar test for binary matched-pairs data: mid-p and asymptotic are better than exact conditional. *BMC Medical Research Methodology*, 13(1):1–8.

[Fellbaum, 1998] Fellbaum, C. (1998). *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, USA.

[Finkelstein et al., 2002] Finkelstein, L., Gabrilovich, E., Matias, Y., Rivlin, E., Solan, Z., Wolfman, G., and Ruppin, E. (2002). Placing search in context: The concept revisited. *ACM Transactions on Information Systems*, 20(1):116–131.

[Firth, 1957] Firth, J. R. (1957). A synopsis of linguistic theory 1930-55. *Studies in Linguistic Analysis (special volume of the Philological Society)*, 1952-59:1–32.

[Florian et al., 2002] Florian, R., Cucerzan, S., Schafer, C., and Yarowsky, D. (2002). Combining classifiers for word sense disambiguation. *Natural Language Engineering*, 8(4):327–341.

[Goldberg and Levy, 2014] Goldberg, Y. and Levy, O. (2014). word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method. *CoRR*, abs/1402.3722.

[Guo et al., 2014] Guo, J., Che, W., Wang, H., and Liu, T. (2014). Learning sense-specific word embeddings by exploiting bilingual resources. In *Proceedings of 25th International Conference on Computational Linguistics (COLING 2014)*, pages 497–507, Dublin, Ireland.

[Hamp and Feldweg, 1997] Hamp, B. and Feldweg, H. (1997). GermaNet – a lexical-semantic net for German. In *Proceedings of the ACL workshop on Automatic Information Extraction and Building of Lexical Semantic Resources for NLP Applications*, pages 9–15, Madrid, Spain.

[Hope and Keller, 2013a] Hope, D. and Keller, B. (2013a). MaxMax: a graph-based soft clustering algorithm applied to word sense induction. In *Proceedings of the 14th International Conference on Computational Linguistics and Intelligent Text Processing - Volume Part I*, pages 9–15, Samos, Greece.

[Hope and Keller, 2013b] Hope, D. and Keller, B. (2013b). UoS: a graph-based system for graded word sense induction. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 689–694, Atlanta, GA, USA.

[Hovy et al., 2006] Hovy, E., Marcus, M., Palmer, M., Ramshaw, L., and Weischedel, R. (2006). OntoNotes: the 90% solution. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 57–60, New York City, NY, USA.

[Huang et al., 2012] Huang, E. H., Socher, R., Manning, C. D., and Ng, A. Y. (2012). Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 873–882, Jeju Island, Korea.

[Ide et al., 2004] Ide, N., , Ide, N., and Suderman, K. (2004). The american national corpus first release. In *Proceedings of the Fourth Language Resources and Evaluation Conference (LREC 2004)*, pages 1681–1684, Lisbon, Portugal.

[Jurgens, 2013] Jurgens, D. (2013). Embracing ambiguity: A comparison of annotation methodologies for crowdsourcing word sense labels. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 556–562, Atlanta, GA, USA.

[Jurgens and Klapaftis, 2013] Jurgens, D. and Klapaftis, I. (2013). SemEval-2013 Task 13: Word sense induction for graded and non-graded senses. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 290–299, Atlanta, GA, USA.

[Klein and Manning, 2003] Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1 (ACL 2003)*, pages 423–430, Sapporo, Japan.

[Klein et al., 2002] Klein, D., Toutanova, K., Ilhan, H. T., Kamvar, S. D., and Manning, C. D. (2002). Combining heterogeneous classifiers for word-sense disambiguation. In *Proceedings of the ACL-02 workshop on Word Sense Disambiguation: Recent successes and future directions*, pages 74–80, Philadelphia, PA, USA.

[Kumar and Vassilvitskii, 2010] Kumar, R. and Vassilvitskii, S. (2010). Generalized distances between rankings. In *Proceedings of the 19th International Conference on World Wide Web (WWW 2010)*, pages 571–580, Raleigh, North Carolina, USA.

[Lancichinetti et al., 2009] Lancichinetti, A., Fortunato, S., and Kertész, J. (2009). Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015.

[Lau et al., 2013] Lau, J. H., Cook, P., and Baldwin, T. (2013). unimelb: Topic modelling-based word sense induction. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 307–311, Atlanta, GA, USA.

[Leacock et al., 1998] Leacock, C., Miller, G. A., and Chodorow, M. (1998). Using corpus statistics and WordNet relations for sense identification. *Computational Linguistics*, 24(1):147–165.

[Leacock et al., 1993] Leacock, C., Towell, G., and Voorhees, E. (1993). Corpus-based statistical sense resolution. In *Proceedings of the Workshop on Human Language Technology (HLT 1993)*, pages 260–265, Princeton, New Jersey, USA.

[Lee and Ng, 2002] Lee, Y. K. and Ng, H. T. (2002). An empirical evaluation of knowledge sources and learning algorithms for word sense disambiguation. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 41–48, Philadelphia, PA, USA.

[Lesk, 1986] Lesk, M. (1986). Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone. In *Proceedings of the 5th Annual International Conference on Systems Documentation (SIGDOC 1986)*, pages 24–26, Toronto, Ontario, Canada.

[Levy and Goldberg,] Levy, O. and Goldberg, Y. Neural word embedding as implicit matrix factorization. In *Proceedings of Annual Conference on Neural Information Processing Systems (NIPS 2014)*.

[Levy and Goldberg, 2014a] Levy, O. and Goldberg, Y. (2014a). Dependency-based word embeddings. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, Volume 2: Short Papers, (ACL 2014)*, pages 302–308, Baltimore, MD, USA.

[Levy and Goldberg, 2014b] Levy, O. and Goldberg, Y. (2014b). Linguistic regularities in sparse and explicit word representations. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, Baltimore, MD, USA.

[Levy et al., 2015] Levy, O., Goldberg, Y., and Dagan, I. (2015). Improving distributional similarity with lessons learned from word embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

[Li and Jurafsky, 2015] Li, J. and Jurafsky, D. (2015). Do multi-sense embeddings improve natural language understanding? *CoRR*, abs/1506.01070.

[Lin, 1998] Lin, D. (1998). An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning (ICML 1998)*, volume 98, pages 296–304, Madison, WI, USA.

[Liu et al., 2015] Liu, Y., Liu, Z., Chua, T.-S., and Sun, M. (2015). Topical word embeddings. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 2418–2424, Austin, TX, USA.

[Manandhar and Klapaftis, 2009] Manandhar, S. and Klapaftis, I. P. (2009). SemEval-2010 Task 14: evaluation setting for word sense induction & disambiguation systems. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, pages 63–68, Boulder, CO, USA.

[McNemar, 1947] McNemar, Q. (1947). Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

[Mikolov et al., 2010] Mikolov, T., Karafiát, M., Burget, L., Cernockỳ, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In *Proceedings of the 11th Annual Conference of the International Speech Communication Association (INTERSPEECH 2010)*, pages 1045–1048, Makuhari, Chiba, Japan.

[Mikolov et al., 2013b] Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168.

[Mikolov et al., 2013c] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. In *Proceedings of the 27th Annual Conference on Neural Information Processing Systems 2013*, pages 3111–3119, Lake Tahoe, NV, USA.

[Mikolov et al., 2013d] Mikolov, T., Yih, W., and Zweig, G. (2013d). Linguistic regularities in continuous space word representations. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, GA, USA.

[Miller et al.,] Miller, T., Biemann, C., Zesch, T., and Gurevych, I. Using distributional similarity for lexical expansion in knowledge-based word sense disambiguation. pages 1781–1796.

[Moffat and Zobel, 2008] Moffat, A. and Zobel, J. (2008). Rank-biased precision for measurement of retrieval effectiveness. *ACM Transactions on Information Systems*, 27(1):2:1–2:27.

[Mooney, 1996] Mooney, R. J. (1996). Comparative experiments on disambiguating word senses: An illustration of the role of bias in machine learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 1996)*, pages 82–91, Philadelphia, PA, USA.

[Morin and Bengio, 2005] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics (AISTATS 2005)*, pages 246–252, Bridgetown, Barbados.

[Moro et al., 2014] Moro, A., Raganato, A., and Navigli, R. (2014). Entity linking meets word sense disambiguation: a unified approach. *Transactions of the Association for Computational Linguistics*, 2:231–244.

[Navigli, 2009] Navigli, R. (2009). Word sense disambiguation: A survey. *ACM Computing Surveys*, 41(2):10:1–10:69.

[Navigli and Ponzetto, 2012] Navigli, R. and Ponzetto, S. P. (2012). BabelNet: the automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217–250.

[Neelakantan et al., 2014] Neelakantan, A., Shankar, J., Passos, A., and McCallum, A. (2014). Efficient non-parametric estimation of multiple embeddings per word in vector space. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1059–1069, Doha, Qatar.

[Ng, 1997a] Ng, H. T. (1997a). Exemplar-based word sense disambiguation: Some recent improvements. *CoRR*, cmp-lg/9706010.

[Ng, 1997b] Ng, H. T. (1997b). Getting serious about word sense disambiguation. In *Proceedings of the ACL SIGLEX Workshop on Tagging Text with Lexical Semantics: Why, What, and How*, pages 1–7, Washington, D.C., USA.

[Nieto Piña and Johansson, 2016] Nieto Piña, L. and Johansson, R. (2016). Embedding senses for efficient graph-based word sense disambiguation. In *Proceedings of the Human Language Technology Conference of the NAACL (to appear)*, San Diego, CA, United States.

[Nivre et al., 2006] Nivre, J., Hall, J., and Nilsson, J. (2006). MaltParser: a data-driven parser-generator for dependency parsing. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*.

[Panchenko, 2013] Panchenko, A. (2013). *Similarity measures for semantic relation extraction*. PhD thesis, Université catholique de Louvain, Louvain-la-Neuve, Belgium.

[Pantel and Lin, 2002] Pantel, P. and Lin, D. (2002). Discovering word senses from text. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 613–619, Edmonton, AB, Canada.

[Passonneau et al., 2012] Passonneau, R. J., Bhardwaj, V., Salleb-Aouissi, A., and Ide, N. (2012). Multiplicity and word sense: evaluating and learning from multiply labeled word sense annotations. *Language Resources and Evaluation*, 46(2):219–252.

[Passos et al., 2014] Passos, A., Kumar, V., and McCallum, A. (2014). Lexicon infused phrase embeddings for named entity resolution. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning (CoNLL 2014)*, pages 78–86, Baltimore, MD, USA.

[Pedersen, 2007] Pedersen, T. (2007). *Learning Probabilistic Models of Word Sense Disambiguation*. PhD thesis, Southern Methodist University, Dallas, TX, USA.

[Pedersen and Bruce, 1997] Pedersen, T. and Bruce, R. (1997). Distinguishing word senses in untagged text. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*, pages 197–207, Providence, RI, USA.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). Glove: global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar.

[Qiu et al., 2014] Qiu, L., Cao, Y., Nie, Z., Yu, Y., and Rui, Y. (2014). Learning word representation considering proximity and ambiguity. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, pages 1572–1578, Québec City, Québec, Canada.

[Rajaraman and Ullman, 2011] Rajaraman, A. and Ullman, J. D. (2011). *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA.

[Řehůřek and Sojka, 2010] Řehůřek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of the Language Resources and Evaluation Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta.

[Reisinger and Mooney, 2010] Reisinger, J. and Mooney, R. J. (2010). Multi-prototype vector-space models of word meaning. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 109–117, Los Angeles, CA, USA.

[Rong, 2014] Rong, X. (2014). word2vec parameter learning explained. *CoRR*, abs/1411.2738.

[Rothe and Schütze, 2015] Rothe, S. and Schütze, H. (2015). AutoExtend: extending word embeddings to embeddings for synsets and lexemes. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, Volume 1: Long Papers*, pages 1793–1803, Beijing, China.

[Schütze, 1998] Schütze, H. (1998). Automatic word sense discrimination. *Computational linguistics*, 24(1):97–123.

[Tian et al., 2014] Tian, F., Dai, H., Bian, J., Gao, B., Zhang, R., Chen, E., and Liu, T.-Y. (2014). A probabilistic model for learning multi-prototype word embeddings. In *Proceedings of the 25th International Conference on Computational Linguistics: Technical Papers (COLING 2014)*, pages 151–160, Dublin, Ireland.

[Towell and Voorhees, 1998] Towell, G. G. and Voorhees, E. M. (1998). Disambiguating highly ambiguous words. *Computational Linguistics*, 24(1):125–145.

[Turian et al., 2010] Turian, J., Ratinov, L., and Bengio, Y. (2010). Word representations: A simple and general method for semi-supervised learning. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010)*, pages 384–394, Uppsala, Sweden.

[van Dongen, 2000] van Dongen, S. (2000). *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, Utrecht, Netherlands.

[Vasilescu et al., 2004] Vasilescu, F., Langlais, P., and Lapalme, G. (2004). Evaluating variants of the Lesk approach for disambiguating words. In *Proceedings of Language Resources and Evaluation Conference (LREC 2004)*, pages 633–636, Lisbon, Portugal.

[Véronis, 2004] Véronis, J. (2004). HyperLex: Lexical cartography for information retrieval. *Computer Speech and Language*, 18(3):223–252.

[Widdows and Dorow, 2002] Widdows, D. and Dorow, B. (2002). A graph model for unsupervised lexical acquisition. In *Proceedings of the 19th international conference on Computational linguistics*, pages 1–7, Taipei, Taiwan.

[Wu and Giles, 2015] Wu, Z. and Giles, C. L. (2015). Sense-aware semantic analysis: A multi-prototype word representation model using Wikipedia. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, pages 2188–2194, Austin, TX, USA.

[Zhang et al., 2013] Zhang, Z., Gentile, A. L., and Ciravegna, F. (2013). Recent advances in methods of lexical semantic relatedness – a survey. *Natural Language Engineering*, 19(04):411–479.