Characterizing Word Representations
For Natural Language Processing

Manuel R. Ciosici

PhD Dissertation

Department of Computer Science
Aarhus University
Denmark
Characterizing Word Representations For Natural Language Processing

A Dissertation
Presented to the Faculty of Science and Technology
of Aarhus University
in Partial Fulfillment of the Requirements
for the PhD Degree

by
Manuel R. Ciosici
July 12, 2019
Abstract

Over the last decade, Natural Language Processing (NLP) technologies have entered mainstream usage. Machine Translation, Intelligent Virtual Assistants like Siri and Alexa, clever assistive smartphone keyboards, and hyper-intelligent web search engines like Google have had, and continue to have, a deep impact on modern society. All smart systems working with natural language share one fundamental aspect: they require word representations for natural language that are consistent, compact, contain deep linguistic knowledge, and support resolution of ambiguity inherent in human language. Therefore, research into word representations that encode linguistic and semantic knowledge is still essential.

In this dissertation, I present my work on several important open problems of word representations: understanding what information they contain, faster computation, and intentional use of word representations in downstream tasks such as abbreviation disambiguation. I empirically show that Brown Clusters, a popular kind of word representations, are well suited for unsupervised learning of morphosyntactic information from natural languages, even when presented with small unstructured text corpora. This is based on extensive empirical experimentation with several indo-european languages. My research indicates that Brown Clusters are prime candidates for representing words in downstream NLP tasks that require syntactic knowledge.

Following, based on insights gained through empirical studies of algorithms that compute Brown Clusters, I propose methods to speed up cluster computation. My research shows that using Hybrid Exchange-Brown algorithm can be up to 21 times faster than the original Brown clustering algorithm without requiring specialized hardware.

Using insights into the inner workings of word2vec (another highly popular word representation method), I propose UAD, a state-of-the-art method for resolving the meaning of ambiguous abbreviations based on their usage in context. UAD intentionally uses word representations to support disambiguation of large numbers of abbreviations in the same model, in a completely unsupervised manner. I show that UAD outperforms previous state-of-the-art abbreviation disambiguation methods and can easily be used in new domains and languages as it does not rely on hand-designed, language-specific, features and is an unsupervised method. Furthermore, UAD’s intentional use of word representations results in disambiguation models that support the identification of challenging abbreviation meanings, and corrections without reliance on manually labeled data, or requiring model retraining.
I løbet af det sidste århi har teknologier for Sprogteknologi (Natural Language Processing - NLP) været indført i almindelig brug. Maskinoversættelse, intelligente virtuelle assistenter som Siri og Alexa, intelligente smartphone-tastaturer og hyperintelligente websøgemaskiner som Google både har haft og har fortsat en dyb indvirkning på det moderne samfund. Alle smarte systemer, der arbejder med sprog, deler et grundlæggende aspekt: de kræver repræsentationer af ord, der er konsekvente, kompakte, indeholder dyb sproglig viden og understøtter opklaring af tvetydighed iboende i menneskers sprogbrug. Derfor er forskning i ordrepræsentationer indholdende sproglig og semantisk viden stadig vigtig for sprogteknologier.


På baggrund af indsigter opnået gennem empiriske undersøgelser af algoritmer, der beregner Brown Clusters, foreslår jeg Hybrid Exchange-Brown algoritmen kan beregne klyngerne 21 gange hurtigere end den originale Brown clustering algoritme uden brug af specialiseret hardware.

Acknowledgments

This dissertation was my largest and most complicated project to date and it would not have been possible without help, advice, and support from co-workers, friends, family, and institutions. I am grateful to all of you for your help and confidence.

A massive thank you to Ira Assent, the UNILO guys (Mads Rydahl, Mario Juric, Thomas Laursen) for establishing the project, and not least for believing in me. A special thanks to my adviser, Ira Assent, for guidance, wisdom, listening, and for always being available to help. Thank you to Leon Derczynski and Sean Chester for introducing me to Brown Clusters. I am grateful to my industry supervisors Mario Juric, Mads Villadsen, Søren Andersen, and Hans Lund for taking me under their wings and for being role models.

An industrial PhD is like being part of two families. I would like to thank my Aarhus University Data Intensive family for support and especially to Jan Neerbek and Simon Enni for the countless hours of badminton, camaraderie, and ping-pong of ideas. Big high fives to the entire UNILO family for all the support and coffee breaks filled with discussions and advice. A big thank you to both the AU and UNILO families for putting up with my countless jokes and pranks.

Lastly, I want to thank Ela, Simon, and my parents for listening and being there for me.

Manuel R. Ciosici,
Aarhus, July 12, 2019.
# Contents

Abstract i

Resumé iii

Acknowledgments v

Contents vii

## I Overview

1 Introduction 3

2 Language Models 7
   2.1 Predicting the next token 8
   2.2 Predicting the missing token 12
   2.3 Conclusion 12

3 Word Representations 15
   3.1 Discrete representations 15
   3.2 Continuous Representations 18
   3.3 Challenges and Open Questions 21
   3.4 Conclusion 25

4 Contributions and Future Work 27
   4.1 Quantifying the morphosyntactic content of Brown clusters 27
   4.2 Accelerating computation of Brown clusters 29
   4.3 Unsupervised Abbreviation Disambiguation 31

## II Publications

5 Morphosyntactic content of Brown Clusters 39
   5.1 Introduction 39
   5.2 Background 40
5.3 Metric selection ............................................. 41
5.4 Experiments ............................................... 43
5.5 Conclusion .................................................. 50

6 Accelerated Hierarchical Clustering .................. 53
  6.1 Introduction ............................................... 53
  6.2 Background ............................................... 54
  6.3 Exchange Clustering ...................................... 55
  6.4 Experimental Setup ..................................... 56
  6.5 Results and Analysis ................................... 57
  6.6 Conclusion ................................................ 59

7 Unsupervised Abbreviation Disambiguation ........ 61
  7.1 Introduction ............................................... 62
  7.2 Related Work ............................................. 63
  7.3 Method .................................................... 65
  7.4 Experiments ............................................... 73
  7.5 Conclusion ................................................ 83

8 Abbreviation Expander .................................... 85
  8.1 Introduction ............................................... 85
  8.2 System Architecture .................................... 86
  8.3 Related Work ............................................. 89
  8.4 Conclusion ................................................ 89

9 Abbreviation Explorer ..................................... 91
  9.1 Introduction ............................................... 91
  9.2 Pre-evaluation analysis .................................. 92
  9.3 Evaluation of Abbreviation Explorer .................. 95
  9.4 Conclusion ................................................ 96

Bibliography .................................................. 97
Part I

Overview
Chapter 1

Introduction

The advent of computers and the Internet brought instant access to troves of knowledge, encoded predominantly in unstructured text. Some information comes in the form of digital counterparts to existing physical text-based knowledge containers, such as academic journals, encyclopedias, books, and magazines. Other forms of classical text-based information formats have become a lot more popular due to digitalization, as is the case for online versions of discussion forums, product reviews, and do-it-yourself guides. Yet others are native to the digital environment, such as status updates, tweets and other forms of instant messages. With digitalization, the problem of accessing information stops being characterized by data starvation, but rather by drowning in data. Indeed, most information is contained in an ocean of documents containing unstructured text. Consider an aerospace engineer tasked with improving the fuel efficiency of a new high-speed aircraft. With rapidly advancing research, relying on classical, pre-digital, information search and retrieval techniques like manually created index cards and directories is not possible. Today’s knowledge workers depend on smart search engines and document recommendation systems to identify potentially useful content in collections of documents containing unstructured text. Today’s corpora range in size from small collections of several million academic articles to the continuously increasing World Wide Web.

Search engines, whether web-scale or corpus-specific, are highly complex systems that have evolved from finding and retrieving documents from a large collection, to question answering systems that are expected to learn and understand human language. Besides search engines, intelligent virtual assistants are a new class of systems that take queries from humans in natural language, interpret them, and execute commands, whether that is to schedule meetings, take notes, or to use search engines in order to answer questions. At the base of both search engines and virtual assistants lie representations of natural, human language that encode information about the language’s morphology, syntax, grammar, and about the physical world. For example, when the engineer in our case queries increased fuel efficiency for PCBs, the search engine is expected to understand the intended meaning of the abbreviation PCB, namely Plenum Chamber Burning. But the computer system is also expected
to know that a plenum chamber burner is a kind of afterburner in jet engines and that afterburners are sometimes referred to as reheat systems.

Researchers working with Natural Language Processing have proposed several methods for representing words in ways that allow computers to reason about unstructured text. Earlier word representations simply identified words by unique identifiers. Statistical methods for word representations often use large corpora of text in order to construct representations of words based on their context and co-occurrence with other words in the language. Representations proposed so far range from membership to a cluster of words, vectors of floating-point values assigned to each word, and to vectors constructed for each usage of a word, based on its surrounding context. The proposed methods vary in terms of their underlying assumptions, the amount and kind of information encoded. Some methods assume a one-to-one mapping of words to representations such as cluster IDs or word vectors, while others construct a representation for each instance of a word use. Some methods consider the context of a word in any given sentence to be only the immediately preceding words, others take it to be linear (i.e. the words immediately to the left or right), some use dependency graphs to establish context, and others learn variable contexts based on observations in thousands of sentences. Methods vary also by the amount of information they pay attention to. Some only look at unique words (for example considering singing to be completely different from sing or song), others also take into account character-level information.

While it is easy to enumerate the differences between various word representation methods, it is considerably more difficult to understand their effect on the information encoded in the representations. What kind and how much knowledge is contained in any given word representation is crucial as word representations are at the foundation of most Natural Language Processing (NLP) systems. The stakes in choosing the right word representation method are higher than making a choice that takes more to compute, requires more data, yet yields the same performance as previous methods [8]. In fact, some word representations have been found to have accidentally learned to discriminate the same way humans do [14, 19]. Only by thoroughly studying word representations can we hope to purposefully use them in downstream NLP systems and to avoid (or at least limit and understand) their biases.

In this dissertation, we focus on two widely used word representations and systematically provide insights into their properties. We show that word clusters constructed from two-sided class-based bi-gram language models are highly effective at encoding morphosyntactic information and thus well suited for downstream systems that require such information. We then show how the clusters can be computed faster. Turning to word vectors, we show how the abbreviation disambiguation task can be solved at scale using unsupervised learning by relying exclusively on word representations. Our disambiguation method is distinguished not just by its purposeful use of word representations, performance, and unsupervised nature, but is also transparent and supports inspections of the vector space that allow users to identify issues in data processing or cases where more data is required.
Thesis outline

Part I is structured as follows: This chapter introduces the topic and some open questions. In Chapter 2, I provide an introduction to language models as utilized in Natural Language Processing, discuss weaknesses such as brittleness, and provide an overview of methods that alleviate brittleness. Chapter 3 builds upon the topics introduced in Chapter 2 and introduces word representations, both discrete (Section 3.1) and continuous (Section 3.2). Following, I discuss open questions and challenges faced by word representations in Section 3.3. In Chapter 4, I discuss my contributions and present future research plans in the areas of: accelerating computation of Brown clusters (Section 4.2), understanding the information content of Brown clusters (Section 4.1), and unsupervised abbreviation disambiguation using word vector spaces (Section 4.3).

Part II contains published work, namely Chapters 5 to 9. Only editing related to formatting have been made to the papers reproduced in Part II of this dissertation. More specifically, Part II consists of:


Chapter 7: Manuel R. Ciosici, Tobias Sommer and Ira Assent (2019). Unsupervised Abbreviation Disambiguation Contextual disambiguation using word embeddings [33];


Chapter 2

Language Models

Learning the meaning and role of words in a language is impossible without first understanding the language. Since we cannot teach computers languages the same way we teach humans, we must find other ways computers can learn to understand language. Language models provide computers foundational information about the meaning, order, and syntactic role of words. This chapter provides an overview of language models and serves to establish theoretical grounding for the following chapters.

Language models, as their name implies, are models of language flow. They model transitions between words, but can also rely on characters or phrases [70]. Being models, rather than the object being modeled, language models are (1) not perfect and (2) rely on a number of assumptions. The language models we discuss in this chapter use statistical information about word order, gained observations of word transitions in large corpora, to predict word sequences in natural language text. Predictions are based on contextual information, where some variation can exist in the definition of context.

Models of language have several immediate applications, the most popular of
which is predictive text input [117]. Most smartphones today use some form of predictive text input to make it less of a burden for users to type long text messages or emails. Figure 2.1 shows a modern smartphone keyboard that uses a language model to predict the next utterance in a user’s input. This application of language models in everyday life is such pervasive that it is noticed only when it fails or autocorrects a word to something that causes an embarrassing situation. Text prediction has been with mobile phones for a long time; even feature phones of the 2000s relied on predictive text input for the widely spread T9 input method\footnote{https://en.wikipedia.org/wiki/T9_(predictive_text)}.

Speech recognition is another task that often benefits from predictive language models [6, 51]. As an utterance is translated from sound waves into written words, a predictive language model helps choose the most likely word sequence uttered by the user based on what has been spoken already. Predictive language models help speech recognition systems handle homophones (words with the same pronunciation, but different meanings), for example, new versus knew in the user utterance “She told me everything she ___”.

In machine translation, language models improve performance by (among other things) helping to generate more natural translations [16, 18, 89, 91]. Here, language models can help with automatic alignment between words from different languages [89, 90], or improve the translation of phrases [16, 55].

In the remainder of this chapter, I will discuss language models in detail, including some of their weaknesses. In Chapter 3, I will build on language models to introduce word representations and will discuss how assumptions made in language models affect word representations constructed on top of them.

### 2.1 Predicting the next token

Modeling language can be formalized as the task of predicting words in a sequence of text. In this section, we will focus on predicting the next token \( w_i \) in a sequence of text \( S = w_0, w_1, \ldots w_{|S|-1} \), assuming that we have seen only tokens in the sequence up to the word just before the one to be predicted. A token is a sequence of characters that together convey semantic meaning and is analyzed as a unit, i.e. it is atomic.

Predicting the next token \( w_i \) in a sequence \( S \) is expressed as a probability. Since text in \( S \) is observed as it is produced, the probability \( P \) is conditioned on the already observed tokens:

\[
P(w_i|S) = p(w_i|w_{i-1}, w_{i-2}, \ldots, w_1, w_0) \tag{2.1}
\]

However, conditioning on all observed tokens is practically infeasible as it requires that we observe every possible sequence in a language and estimate probabilities. We must, therefore, make some limiting assumptions. \( N \)-gram language models, as their name implies, use ordered sequences of \( n \) tokens (called n-grams), to shrink the context used for prediction, thus simplifying the condition assumption. In the
2.1. PREDICTING THE NEXT TOKEN

The broken car is repaired. The red and broken car is repaired.
La voiture cassée est réparée. La voiture rouge et cassée est réparée.

(a) (b)

Figure 2.2: An example of how grammatical gender can be inferred using bi-gram language models.

case of n-gram language models, the probability of the next token \( w_i \) in a sequence \( S \) is conditioned only on the n-gram it is the last member of (i.e., the previous \( n - 1 \) tokens) \([70]\):

\[
P(w_i|S) = p(w_i|w_{i-1}, w_{i-2}, \ldots, w_{i-n+1})
\] (2.2)

As n-gram language models model transitions between all possible n-grams in a vocabulary \( V \), they must estimate \( |V|^n \) parameters. Making reliable estimations for so many transition probabilities is challenging as natural language tends to have large vocabularies, often with hundreds of thousands of words \([74, 92]\). On top of this, morphological inflections (number, gender, case, tenses, etc.) increase the perceived vocabulary size to millions of elements. The immediate solution might seem to be the use of very large text corpora for estimating transition probabilities. But, due to the Zipfian distribution of words in natural language \([119]\), most words in the vocabulary will rarely be observed and thus most n-grams will never occur, leading to unreliable transition estimates.

To maintain tractability of n-gram language models and increase the reliability of transition estimates, low values of \( n \) are typically used \([70]\). Using 1-grams leads to weak language models as their output is informed exclusively by the probability distribution of words in a corpus of text and no contextual information is used (i.e., from the observed transitions). Bi-gram models reduce the n-gram language model condition assumption to only the preceding word:

\[
P(w_i|S) = p(w_i|w_{i-1})
\] (2.3)

Using bi-grams facilitates modeling of transitions within the language model. We must estimate transition probabilities between any two tokens in the vocabulary, so only \( |V|^2 \) parameters. Unfortunately, bi-gram language models cannot capture long-distance relationships between words. In Figure 2.2a, a bi-gram language model can easily predict the feminine version cassée instead of the masculine cassé, based on the grammatical gender of the previous token (voiture meaning car is feminine in French). But, in Figure 2.2b, that is no longer possible with a bi-gram language model, as the previous token et (meaning and) does not provide any information about grammatical gender. The immediate solution is to use tri-grams, i.e. condition on the previous two tokens. However, tri-grams can also fail to capture some long-distance information. Following with the example in Figure 2.2, a tri-gram language
model would not be able to properly predict the correct version of *cassé* in the sentence *La voiture n’est pas cassée* as there are three tokens separating the noun *voiture* containing grammatical gender information from the adjective that modified it (i.e., *n’, est*, and *pas*). While one can always find examples of natural language where any given n-gram model fails, in practice bi-gram and tri-gram language models work well and have been widely used [87, 117].

The problem of unreliable transition estimates is essentially one of overfitting. Due to the large vocabulary of natural language, and the high number of transition estimations required, n-gram language models observe only a subset of the possible n-grams [17]. For example, bi-grams observed in personal documents such as chat, manuals, and stories differ considerably from those in newspaper text [117]. This results in predictive models highly tuned to the style and domain of the text corpus used for deriving the language model. When applied to text from a different domain or writing style, such models often exhibit lower performance. This problem is often referred to as language model brittleness [17] and is an instance overfitting. The situation appears when machine-learned models learn to match the probability distributions specific to the sampled training data so well, that they do not function properly on other data that varies slightly due to small changes in probability distributions. In other words, overfitted models do not generalize.

Several approaches have been proposed to address brittleness in language models and most of them can be combined. In the remainder of this section, I will provide a brief overview. **Interpolation and dynamic adaptation** of language models [2, 11, 50, 51] combine a language model with extra information. Interpolation of high n-gram language models with lower n-gram language models [2, 50], using a class-based language model [17] (which we will introduce separately below), or a local language model derived on a task-specific corpus [117]. Interpolation can reduce overfitting by including lower n-gram models, it does require high-performance n-gram models to interpolate with. Dynamic adaptation can help a language model better fit the desired style by placing more weight on the observations in a small, but highly relevant corpus [11, 51].

Another option is the use of **skip-grams** in order to extract more n-grams from the available corpus by allowing each n-gram to skip up to *k* tokens [48]. This can be applied to size n-gram by relaxing the requirement that members of the n-gram are immediate neighbors in the text sequence. *k*-skip-*n*-grams are n-grams where up to *k* tokens have been skipped while forming the n-gram. Some example *k*-skip-*n*-grams from the English language sentence in Figure 2.2b are: *{The, and}* (a 1-skip-bi-gram since it skips the word *red*), *{and, car, repaired}* (a 2-skip-tri-gram since it skips the tokens *broken* and *is*), or *{and, is, repaired}* (also a 2-skip-tri-gram). For tri-grams and bi-grams, *k*-skip-*n*-gram extraction has been empirically shown to extract n-gram usage information from English corpora that is equivalent to what is possible from corpora 4 times larger if using only equivalent *n*-grams [48]. Thus, they address brittleness by extracting more transition information. However, their effectiveness for
languages other than English has not been established yet.  

Class-based language models [17] are another way to address brittleness. Here, a layer of abstraction is added to n-grams by grouping words into classes and estimating transition probabilities between the n-grams of classes rather than n-grams of words. Thus the probability of observing the word $w_i$ given the previous word $w_{i-1}$ is estimated based on the probability $p_1$ of transitioning from the cluster of the previous word $c(w_{i-1})$, to the cluster $c(w_i)$ of the word $w_i$:

$$P(w_i|w_{i-1}) = p_0(w_i|c(w_i)) \cdot p_1(c(w_i)|c(w_{i-1})) \quad (2.4)$$

Here, $c : V \rightarrow C$ is a function returning the class of each word $w_i \in V$. It is a many-to-one mapping that enforces a hard clustering of words, i.e. each word can be in one and only one cluster. The language model must then both estimate transition probabilities between classes (often referred to as clusters) and learn a function $c$ that maps words from the vocabulary $V$ into classes $C$. The outcome of the abstraction provided by the clusters is a loss in prediction accuracy, but increased stability over different corpus domains and styles [17]. The increase in stability comes from the reduced number of parameters. As opposed to a bi-gram language model that requires $|V|^2$ parameter estimations, a two-sided class-based language model only requires $|C|^2 + |V|$ parameter estimations. The first $|C|^2$ parameters estimate transitions between the clusters, while the remaining $|V|$ model emissions from the clusters. The number of clusters used is usually in the range of hundreds to a few thousand [17, 53, 72], which makes the model considerably smaller.

The class-based language model formalized in Equation (2.4) is referred to as a two-sided class-based language model (classes/clusters appear on both sides of $p_1$). This is to differentiate from single-sided bi-gram class-based language models [36, 37, 120] which condition $p_1$ directly on the previous word $w_{i-1}$:

$$P(w_i|w_{i-1}) = p_0(w_i|c(w_i)) \cdot p_1(c(w_i)|w_{i-1}) \quad (2.5)$$

Single-sided class-based language models are theoretically more powerful as they can model more kinds of transitions, but that comes at the cost of more parameter estimations, more specifically $|V| \cdot |C| + |V|$. Class-based language models have been proposed using bi-grams [17, 53], or tri-grams [72]. They can, in fact, be expressed on top of n-grams or skip-grams (which we introduced above), although no public implementations exist yet. They can be used by themselves, combined with higher n-gram models via interpolation (introduced above) [17], or adapted using task-specific corpora via dynamic adaptation (also introduced above).

\[^2\text{word2vec, a popular method for constructing word representations that we discuss in Chapter 3, relies on k-skip-bi-grams and has been widely successful in a number of languages.}\]
2.2 Predicting the missing token

Predicting the next token in a sequence of text is not the only kind of predictive task that can be modeled using a language model. Instead, we may want to predict a missing token, given context on both sides of the token, similar to word guessing games. For example, in the Romanian sentence *In magazin, cel mai _____ telefon este prea scump*\(^3\), one can predict that the missing word must be an adjective, in the singular, in the masculine form as it modifies the singular, masculine noun *telefon* meaning phone, and it must be an adjective that can be applied to the word telephone. Good candidates are *nou* (new) or *avansat* (advanced), but not *adormit* (sleepy). Thus, formally the probability of a word \(w_i\) in a sequence \(S\) is conditioned on everything in the sequence, except the word itself:

\[
P(w_i|S) = p(w_i|S \setminus \{w_i\})
\] (2.6)

This kind of “find the missing word” task can be formulated in different ways. As with predictive language models, we cannot condition on the entire sequence \(S\) as it is impossible to observe all possible token sequences in a language. Thus, a more practical version limits the context size:

\[
P(w_i|S) = p(w_i|w_{i-c}, w_{i-c+1} \ldots w_{i-1}, w_{i+1}, w_{i+2}, \ldots w_{i+c})
\] (2.7)

In Equation (2.7), the context consists of the token sequence containing tokens up to a distance of \(c\) to both sides of \(w_i\). Unfortunately, this kind of language model requires the same number of parameter estimations as an \(n\)-gram language model where \(n = c\), i.e. \(|V|^c\) parameter estimations. If we change the list of tokens to be an unordered list of \(c\) tokens on both sides, we get something similar to the Continuous Bag of Words (CBoW) language model used in *word2vec* \([75]\).\(^4\) Limiting the context to only one token in the set of \(n\) tokens on either side results in the Skip-gram language model used of *word2vec* \([75]\)\(^5\). Multiple other tweaks can be made to the definition of context such as removing tokens not directly connected to the predicted token in a dependency tree \([62]\), keeping track of token order \([65]\), or using separate language models predicting from left-to-right and right-to-left \([97]\).

2.3 Conclusion

In this chapter, I have introduced language models which are widely spread in today’s technology from predictive text input in smartphones \([117]\) to assisting speech recognition systems \([6, 51]\). I presented two tasks on which language models can be

\(^3\)Translation: *In store, the most _____ phone is too expensive*

\(^4\)I will discuss *word2vec* in Chapter 3.

\(^5\)Following the terminology presented in the previous section, *word2vec* uses *c-skip-bi-grams* where \(c\) is the context size. In practice, the NLP community refers to these as skip-grams.
derived: predicting the next token in a sequence of text and filling in the missing word in a sentence. I discussed how using more contextual information via higher n-grams helps models express long-distance relationships and how large numbers of parameter estimations can make language models overfit their training data and become brittle. Following, I provided an overview of methods for addressing overfitting, one of which was class-based language models. In the next chapter, I will discuss another exciting application of language models, namely building word representations.
Chapter 3

Word Representations

Word representations support downstream Natural Language Processing tasks by providing information about word morphology, syntax, and semantics in a way that can be understood by downstream algorithms. This chapter is an introduction to the different kinds of word representations proposed in the literature, how they compare to each other, and the current state of research and understanding of each method.

3.1 Discrete representations

Word clusters (or word classes, as they are sometimes referred to) are partitions of a vocabulary which are expected to be internally coherent morphologically, syntactically, or semantically. They differ from clusters on other kinds of data by the fact that word clusters are generally created based on word usage in a large corpus of text, rather than on features representing each object. They can be flat [53, 61] or hierarchical [17]. The affiliation of a word to a cluster provides meaningful representation to algorithms processing natural language. The expressive power of a set of clusters (or clustering) is determined by the number of partitions the vocabulary was split into, the nature of the clustering (hard or soft), and the algorithm employed for cluster derivation.

There are multiple ways to construct word representations from word clusters, depending on the nature of the clustering. The simplest way is to replace each word with the identifier of the cluster it belongs to. Such a representation allows for $k$ degrees of distinction between words, where $k$ is the number of available clusters. Since each cluster is expected to be coherent from the point of view of morphology, syntax, or semantics, representing each word by its cluster allows algorithms to use information about word relatedness, similarity, or syntactical role. This method does not represent relationships between clusters but is compatible with both hierarchical and flat clusterings [25].

Hierarchies of clusters containing $k$ leaves, where each leaf is a separate cluster, can be represented in multiple ways. A string can be used to describe the path from root to each leaf; and words can be represented by the path to the cluster they are a
CHAPTER 3. WORD REPRESENTATIONS

member of [57]. While this method also supports only $k$ degrees of distinction, the string path is expected to encode relationships between clusters. If algorithm-specific information is stored during cluster derivation, then roll-up can be applied [38] where the history of cluster merges is used to create flat or hierarchical clusterings from algorithm runs with higher values of hyper-parameters. We will return to roll-up in the next section when we discuss Brown clusters.

Brown and Exchange clusters

Brown clusters [17] are a hard, hierarchical partitioning of vocabulary derived during the training of a class-based, two-sided, bi-gram language model as described in Equation (2.4). The algorithm follows a bottom-up approach, by first allocating each word to its own cluster, and then progressively merging the two clusters that lead to the lowest loss in Average Mutual Information (AMI), its optimization goal. AMI is a proxy for the log-likelihood of the underlying class-based bi-gram language model [72]. In other words, the algorithm merges classes to arrive at a language model with $k$ classes under which the likelihood of the training corpus is maximized. Due to the intractability of the optimization problem, the algorithm is both greedy (it picks the best merge at each step) and an approximation (all decisions are made in an active window generally containing the top $k + 1$ most frequent clusters). The active window was set to $k + 1$ by Brown et al. [17] as a way to reduce computation costs. Separating the active window is possible and usually leads to higher performance in downstream tasks [38]. Setting the active window to match the vocabulary size $|V|$ results in the non-approximated greedy algorithm. If the algorithm is stopped when there are no more clusters to include in the window (i.e., all words have been assigned to one of the $k$ clusters), then the clustering is a flat one. However, if the algorithm is run for $k - 1$ steps more, then a hierarchy is constructed on top of the $k$ clusters.

The Exchange clustering algorithm [53] is an alternative to Brown clustering that optimizes for the same optimization goal [72], but takes an iterative approach yielding only a flat clustering. Exchange starts by assigning each word to one of $k$ clusters. Usually, this is performed by assigning the $k - 1$ most frequent words each to their own cluster and the remaining words to the last cluster $k$. For a number of iterations $i$, Exchange considers transplanting each word from the vocabulary $w_i \in V$ from its current cluster to each of the other possible clusters. The word is then moved to that cluster that results in the highest improvement in Average Mutual Information. If no other cluster results in an improvement to AMI, then word remains assigned to its current cluster. Exchange can be labeled as an anytime clustering algorithm as a valid clustering exists throughout the running time of the algorithm. However, just as Brown, Exchange is a greedy algorithm since in every step the effects of every word transplant are considered individually. The algorithm performs no look-ahead or backtracking.

Suster and van Noord [114] propose a version of Brown clustering that uses dependency relations between words in a sentence to construct pairs of words that re-
place bigrams in the original algorithm formulation. Dependency relations [35] connect words in sentences based on syntax. For example, in the sentence *Red, green, and yellow bell peppers are delicious*, there is an *amod* (adjectival modifier) relationship between each of the words denoting colors and the noun *peppers* as well as an *nn* (noun compound modifier) relationship between *bell* and *peppers*. Suster and van Noord [114] propose using pairs of words that are connected, regardless of the geographic distance between the words in a sentence. The same idea is applied in word2vecf, a variant of word2vec discussed in Section 3.2. The dependencies-based variant of Brown seems to improve the performance of clusters as representations for word similarity in Dutch. However, more extensive evaluation covering multiple languages has not yet been explored. Encoding dependency relations as pairs of words is not the only way to include more information for Brown and Exchange. In fact, Martin et al. [72] propose a version of Exchange that uses tri-grams for the same reasons tri-gram language models were proposed initially (see Section 2.1), larger *n-grams* capture longer distance relationships. Since both Brown and Exchange have the same optimization goal, they could both use tri-grams or any *n*-grams. However, no implementation currently exists nor empirical or theoretical evaluations in publications that cover the benefits and drawbacks of such extensions except for the few prediction experiments of Martin et al. [72] which provide some insights into language prediction, but nothing on the representation power of tri-gram-based word clusters.

Using flat clusterings (obtained with Brown or Exchange), words can be represented by the IDs of the clusters they belong to. This kind of word representation includes a layer of abstraction based on the morphosyntactic and semantic coherence of the clusters themselves. If a hierarchical clustering is computed, then multiple options exist for representing words. Instead of using cluster IDs based on integers, we can use bitstrings describing the path to each cluster from the root of the hierarchy [57]. This kind of representation has all the advantages of representing words based on a flat hierarchy but also takes into account cluster relationships. Another option is to use shearing, where prefixes of the bit string at multiple lengths are used to simulate multiple flat clusterings. An alternative to shearing is roll-up [38] where during the hierarchical clustering phase, cluster merges and their order is recorded so that they can be used to computer higher levels of the hierarchy without having to rerun the clustering algorithm, or use clusterings that have never occurred, as with shearing. Roll-up has been shown to improve performance in Named Entity Recognition [38].

One of the biggest drawbacks to word clusters is their computation time. The non-windowed greedy Brown algorithm has a time complexity of $O(|V|^{5})$, while the windowed version runs in $O((|V| - k)k^{2} + k \log k)$ [17]. Exchange runs in $O(k^{2}|V|)$. Thus, computing word clusters for large vocabularies is a time-consuming task. Unfortunately, the algorithms do not allow for much parallelization of work either. While it is possible to parallelize some computations when considering clusters to merge or what clusters to move words to. It is not possible to merge multiple clusters in parallel or move several words at a time.
Word clusters derived using the Brown clustering algorithm have found wide use as word representations in Natural Language Processing (NLP) tasks among which we mention Chinese Word Segmentation [64], Dependency Parsing [8, 57], Named Entity Recognition [39, 64, 116], and Part of Speech tagging [39, 94, 116].

**Other clustering techniques**

Besides clusters constructed on top of the two-sided class-based language model described in the previous section, a number of other clustering algorithms exist. We will review the major types in this section.

Relaxing the two-sided constraint in Equation (2.4) like we did in Equation (2.5) leads to a one-sided bi-gram class-based language model which can also be used to compute word clusters [120]. As discussed in Section 2.1, the one-sided class-based formulation increases the number of language model parameters from \(|C|^2 + |V|\) to \(|V| \ast |C| + |V|\). However, the relaxation provides some avenues for optimization of the clustering algorithm due to the smaller number of updates required at every step since the cluster co-occurrence matrix becomes a word-to-cluster matrix. This has been used by Uszkoreit and Brants [120] to implement a distributed version of Exchange using the Map-Reduce paradigm. It has also been used to speed up clustering on a single computer [36, 37].

Another way to compute word clusters is by using algorithms such as **k-means** [125] on word vector spaces such as those computed by the methods discussed in Section 3.2. These clusters are usually flat and can be used for the same kinds of tasks as Brown clusters discussed earlier, for example Dependency Parsing [8].

### 3.2 Continuous Representations

Word vectors are a popular word representation alternative to word clusters. Traditionally, these contained values from hand-designed features such as unigram frequency, word collocations, prefixes, etc. More recently language models are used in order to derive word vectors [12, 75, 97].

**Methods based on shallow neural networks**

Bengio et al. [12] propose a multi-layer neural network language model that implements predictive n-gram models as defined in Equation (2.2). The neural network is trained to predict the next token in the sequence by taking as input a context consisting of concatenated vectors corresponding to the previous \(n-1\) tokens. The resulting neural network language model has a lower perplexity on English text than tri-gram and four-gram language models and even outperforms Brown clusters [12]. Because the context for each prediction is constructed by concatenating vectors, the neural network is aware of word order. The vectors assigned to each word and tuned as part of the learning process can be used to represent words. Unlike Brown clusters
(Section 3.1) computations for this neural network language model can be easily parallelized at training time which makes the method easier to use on larger amounts of data. In later work, Morin and Bengio [81] propose ways to improve training speed by using a hierarchical architecture and knowledge bases like WordNet.\footnote{The architecture described in this paragraph can easily support multiple hidden layers, which Bengio et al. [12] discuss in detail. However, they decide to perform experiments using a single hidden layer. Because of this, and the influence this work has had on word2vec, I chose to discuss the work of Bengio et al. here, rather than Section 3.2 which is dedicated to methods based on deep neural networks.}

Mikolov et al. [75] follow on the work of Bengio et al. [12] and propose word2vec, a single-layer neural network that implements a language model that predicts the missing word in a sentence using context on both sides of the token, as described in Equation (2.6). The two main contributions made by Mikolov et al. are the formulation of the problem using Continuous Bag of Words (CBOW) and Skip-gram, and a simpler, faster method to derive the language model. In the CBOW model, just as in the Bengio et al. [12] model, a single vector is constructed to represent the context. But, in CBOW the context vector covers both sides of the missing word and is constructed by averaging the vectors of surrounding words instead of being a concatenation of vectors of surrounding words. This formulation reduces the number of parameters in the neural network and simplifies the model by not encoding positional information. To this model, contexts such as The waiter brought a ____ to the client and The client brought a ____ to the waiter are identical. At training time, the error assigned to the context vector is equally propagated to each of the constituent vectors. In the previous example, that means the vector for waiter will be adjusted by the same amount as the less informative vector for The.

The Skip-gram model uses k-skip-bi-grams as described in Chapter 2, and formulates the prediction task as a classification problem. Given a pair $w_i, w_j$ where $j \in \{w_{i-c}, \ldots, w_{i-1}, w_{i+1}, \ldots, w_{i+c}\}$ and $c$ is the context window size, the model is expected to output 1 for those pairs that appear in the corpus being modeled. In this classification formulation, the observed pairs of words represent positive examples. In order to obtain negative examples, word2vec creates unobserved pairs by sampling words from the vocabulary based on their frequency. This is termed negative sampling. Both the Skip-gram and the CBOW models can benefit from negative sampling. For the CBOW model, negative sampling is implemented by replacing the target token with one sampled from the vocabulary based on its frequency distribution.

The vector space resulting from word2vec appears to encode morphological, syntactical, and semantic information as empirically shown in Mikolov et al. [77]. We will discuss the question of what information is encoded in the vector space in Section 3.3 later in this chapter. Vectors resulting from word2vec have been widely used as word representations in many downstream NLP tasks among which Named Entity Recognition [7, 98, 109], Part of Speech tagging [98], and Chunking [98]. They can be used by themselves [98, 109], or as clusters computed over the vectors using algorithms like k-means as briefly mentioned in Section 3.1, for example, as features for dependency parsing [8], or Named Entity Recognition [7].
CHAPTER 3. WORD REPRESENTATIONS

GloVe [96] is a word vector representation method that is often considered a drop-in replacement for word2vec. It constructs word vectors by learning a representation that compresses a word co-occurrence matrix which allows highly efficient implementations. GloVe has been shown to match or outperform alternatives such as word2vec in word analogy tasks [7, 96].

All word representation models presented so far consider words as atomic units. In fact, they all work at the token level. However, information about the meaning and syntactic role is often encoded using morphology. Methods that assume words/tokens are atomic units ignore such information which forces them to learn exclusively from word usage, which might be less efficient. For example, it is possible to learn exclusively from word usage that *undeniable* is in the same relationship to *deniable* as *uncertain* to *certain*. However, that relationship might be inferred more easily if we observe that the two words share the same prefix, namely *un-*. FastText [13] uses this idea retrofitted to the Skip-gram model of word2vec in order to take advantage of subword information in the form of character n-grams. Words are represented as a bag of character n-grams and the Skip-gram model of word2vec is used to learn representations for each character n-gram. Besides allowing the model to learn from subword information, FastText’s representation of words addresses the problem of representing words that are not observed in the training corpus, so-called out-of-vocabulary (OOV) words. There are fewer possible character n-grams than there are word n-grams and since words are represented as a sum of character n-grams they contain, it is possible to represent any possible word in a language once one has learned representations for all possible character n-grams. This is a major advantage over word2vec discussed earlier in this chapter which lacks this ability and usually maps all out-of-vocabulary words to a set random vector.

Language Model Representations using deep neural networks

More recently, continuous word representations are constructed using deep neural networks [40, 97]. While all methods in the previous section can be adapted to deep neural networks, the methods presented here are designed with deep neural architectures in mind.

Embeddings from Language Models (ELMo) [97] use deep neural networks to construct word representations. A deep Recurrent Neural Network (RNN) is used to simultaneously train two language models over the input text, going in opposite directions to each other. The left-to-right language model follows the definition provided in Equation (2.1), while the right-to-left can be viewed as implementing the same equation, but on the reverse of the input string. Using two language models allows ELMo to use both left- and right-side contexts for word prediction, effectively implementing a language model that fills in the missing word, similar to the formulation in Equation (2.7). The main difference between ELMo’s model and Equation (2.7) is that the context is not limited to the immediate words surrounding the word to be filled in, instead it extends up to the end of the sentence and the Recurrent
Neural Network learns what tokens to pay attention to by propagating errors during training.

The neural networks share the same token representations and the same upper softmax layer. Word representations are constructed by using a linear combination of all intermediate layer representations resulting from the two neural networks. Parameters for the linear combination are learned for each downstream task. Thus ELMo does not use static vectors to represent words (i.e., there is no static 1-to-1 mapping between words in the vocabulary and vector representations like with word2vec or GLoVe). Word vector representations are constructed for each task based on the context provided for the word.

Using bi-directional language models gives ELMo a number of advantages over other representation methods. As opposed to word2vec’s CBOW and Skip-gram, ELMo uses word order information through the RNN used by the two language models. The RNN also allows ELMo to encode long-distance relationships, which is not possible with n-gram class-based language models, or with the model proposed by Bengio et al. [12]. Another differentiating factor from these two kinds of representations, and an element shared with word2vec is ELMo’s ability to use context on both sides of the token to be predicted. In contrast to word2vec, ELMo does not use a linear window, and thus it can use information from long-distance relationships without turning to potentially noisy external signals like dependency relations that are used in word2vec [62] or some variants of Brown clustering discussed earlier [114]. A newer alternative to ELMo, BERT [40] uses deep neural networks based on a transformer to learn word representations. The reliance on transformers avoids some of ELMo’s drawbacks that are specific to RNNs, like vanishing and exploding gradients.

3.3 Challenges and Open Questions

Constructing word representations is a challenging task. In this section, I present some aspects that contribute to the task’s difficulty.

Amount of required data

All word representation methods that use statistical information via language models require a certain amount of observations for reliable parameter estimation, as discussed in Chapter 2. Most methods presented use unsupervised learning by training a language model on unstructured text which means that effort associated with manual labeling is not a concern. For popular languages, such as English, general-purpose large corpora can easily be found [23, 122]. However, for less popular languages, or when the downstream task requires domain-specific representations one must consider the data size requirements of word representation methods [98]. As mentioned earlier, except for FastText, shallow methods presented so far use data inefficiently as they ignore character-level information which means that more data is necessary in order to learn from word usage what could be learned from morphology.
Qu et al. [98] attempt to answer the question of training corpus size for various shallow word representation methods by studying four sequence labeling tasks: Part of Speech tagging, Chunking, Named Entity Recognition (NER), and identification of Multi-word Expressions (MWE). Most of the word representation methods discussed in this chapter are included in the evaluation: Brown clusters, word2vec (CBOW and Skip-gram), and GloVe. While word vectors lead to the best performance, for most of the tasks studied, Brown clusters achieve better performance on small corpora.

Among the deep neural network vector representations, methods like ELMo outperform other methods like CoVe biLSTM due to the more efficient use of data [97]. Some tasks, such as identification of Multi-Word Expressions and word similarity benefit from training with task-specific corpora [52, 98].

**Computation time and parallelism**

Word clusters derived from class-based language models have had the most computational challenges. Initially, bi-gram Brown clusters were limited to rather small vocabularies and numbers of clusters due to the performance of single-core computers in the 1990s [17]. For tri-gram word cluster models, due to computational restrictions, early experiments limited vocabularies to only the top 20,000 words and avoided experiments with more than 200 clusters [72]. The main challenge to speeding up Brown and Exchange lies in parallelizing the algorithms. Average Mutual Information is based on the cluster-to-cluster interaction matrix. Moving any word from one cluster to another in Exchange, or merging any two clusters in Brown changes the cluster-to-cluster interaction matrix. Moving one word at a time, or performing a single merge at a time can be guaranteed to improve AMI by taking the best solution at every step. However, moving two words at a time, or performing two merges in parallel may not lead to any improvement in AMI (and could even lead to a decrease) as each word move or cluster merge is based on an analysis that considered the other words and clusters as staying fixed. The only algorithmic improvements to the computation of word clusters with class-based language models were achieved on the single-sided class-based language models discussed in Sections 2.1 and 3.1.

Word vectors computed on top of language models usually benefit from the parallelism inherent in neural networks. None the less, a big step in computational performance was the introduction of word2vec [75] which reduced the computational needs of models like that of Bengio et al. [12] by eliminating layers in the neural network and disregarding word order. Word vector representation methods based on neural network have benefited massively from improvements in computation hardware in the last few years, especially those gained through the use of Graphics Processing Units (GPUs) and application-specific hardware accelerators like Tensor Processing Units (TPUs). None the less, newer models like ELMo and BERT are so computationally hungry due to their deep neural architectures, that training becomes a problem even on modern GPUs and TPUs [40, 97].
3.3. CHALLENGES AND OPEN QUESTIONS

Evaluation

In this section, we will discuss methods that evaluate word representations in order to understand what kind of information they encode and how that can influence downstream processing of natural language. Evaluation of word representations can generally be split into two kinds: evaluations that focus exclusively on performance in downstream tasks and intrinsic evaluations that usually measure how much knowledge from external knowledge bases is encoded in the word representation.

Early work [12, 17, 53, 72] focuses on language model performance by measuring perplexity (the likelihood of a given natural language text under a language model). This is due to the early focus on downstream applications of the language models in speech recognition and predictive text input. Qualitative evaluation of representations is usually limited to manual (often anecdotal) inspection of, for example, word clusters [17, 53, 72].

Evaluation methods using word similarity rely on manually annotated data sets such as SimLex-999 [49], WS353 [44] or SimVerb-3500 [46]. Such data sets consist of word pairs together with a similarity score that is usually the mean of scores attributed by a number of human annotators. The metrics used can describe either semantic or syntactic similarity. Due to its relatively simple formulation and availability of data sets, this method is often used for intrinsic evaluation of both word vector spaces and word clusters. It has, however, a number of drawbacks [10]. First, the data set must be a large and representative sample of the semantic or syntactic knowledge measured. Second, most data sets are constructed on the assumption that there exists a single notion of similarity. Some data sets conflate synonymy, antonymy, hypernymy, co-hyponymy, meronymy and topical relatedness. While annotation guidelines can improve consistency between annotators, the assumption has deeper implications as word and concept similarity might differ from one domain to another.

Another kind of evaluation uses performance on a word analogy task as e.g. Mikolov et al. [75]. A word vector space is used to infer the missing element in a pair of words when given another pair as reference. For example, given a pair like (France, Paris) the word vector space must support easy resolution of the missing element in the pair (___, Berlin). A linear combination of vectors assigned to the three words provided is expected to provide the correct answer, Germany. The task aims to measure if the word representation method has learned to encode vector space various kinds of relationships. Due to its simplicity, the task can be formulated to evaluate semantic or syntactic information. In the case of syntactic evaluation, the analogy task asks for a solution to syntax-oriented questions such small is to smaller

---

2Synonyms are words that have the same meaning (e.g. hurry and rush). Antonyms have opposite meanings (e.g. slow and fast). Hypernyms are words in an is-a relationship (e.g. puppy is a dog) while co-hyponyms are words that share the same hypernym parent (e.g. both puppy and poodle are dogs). Meronyms are constituent parts of a larger concept which are often used to refer to the larger concept (e.g. we can say that the orchard contains 34 apples, instead of saying 34 apple trees). Finally, words that are topically related are those that belong to the same topic (e.g. lion and savanna both belong to the topic of Africa).
as what is to bigger?. The vector space will support calculating the correct answer big if the comparative of adjectives is encoded in the vectors. Large data sets for both flavors of word analogy are easily available, e.g. Mikolov et al. [75]. One drawback of this evaluation is its tendency to evaluate the immediate neighborhood of vectors and return the closest element [10] which tends to return answers based on topical relatedness and does not guarantee that relationships are actually encoded in the word representation vector space.

Another intrinsic evaluation that aims to measure the amount of semantic information encoded in vector spaces takes the form of outlier detection [20]. In this task, vectors from a word vector space are used to identify the outlier from a group of words. The groups are constructed following a semantic theme such as geographic locations or names of planets. As opposed to word similarity evaluations, outlier detection benefits from not relying on human agreement of similarity value. Another benefit of this task is that it can easily be applied to humans and can provide a consistent baseline of human performance. However, the currently available data sets are too small (only 64 points of measurement in the data set of Camacho-Collados and Navigli [20]) and therefore do not provide a reliable evaluation of information encoded in word vector spaces.

Evaluation on downstream tasks is usually performed in order to support a new word representation method [97], or in order to show how a specific word representation can be used in a downstream task [8, 38, 39, 94, 98, 120]. In fact, for word clusters, evaluation is almost exclusively split into two categories: language model performance in predictive tasks (measuring perplexity or Average Mutual Information) [17, 39, 53, 72], and evaluation via performance in downstream tasks [38, 39, 94, 98, 120]. Unfortunately, such evaluations do not provide actionable information about the capabilities and limitations of the method under evaluation. Most evaluations focused on downstream tasks only provide information relevant to the specific tasks used in the evaluation. Following evaluation in downstream tasks, for a full understanding that could guide intentional use of word representations, each representation method must be evaluated on every kind of downstream task. In some cases, such evaluations can provide guidance of hyper-parameter choices. For example, Bansal et al. [8] provide some guidance on constructing representations from word clusters and on hyper-parameter choices for word2vec when used in similarity and Part of Speech tagging.

Another approach to understanding what information is encoded in word representations is modifying methods to take various linguistic aspects into account at training time. Ling et al. [65] set to increase the amount of syntactic information encoded in word2vec spaces by modifying word2vec to take word position into account. While the resulting network takes twice as long to train, the new word vectors are better suited for syntactic tasks like Part of Speech tagging and dependency parsing. Following the idea that linear contexts might not necessarily be representative for word representations based on language models, some research filters the context using dependency relations [62, 114] in an attempt to reduce contextual noise. The result is a dependency aware version of word2vec (named word2vecf) and a vari-
3.4. **CONCLUSION**

In this chapter, I built on the knowledge of language models introduced in Chapter 2 and presented a number of methods for discrete and continuous word representations using language models. I discussed a number of problems affecting derivation of word representations such as computational complexity and requirements on the amount of data. I emphasized evaluation and understanding of information encoded in word representations and discussed the benefits and drawbacks of various approaches to word representation and evaluation.

Another linguistically informed approach is that of Mrkšić et al. [86] who post-process word2vec spaces by adjusting vectors for synonyms and antonyms in order to improve performance in word similarity tasks. In a slightly different approach, Kiela et al. [52] distinguish between similarity and relatedness. During training, word vectors are “nudged” to represent similarity based on an external thesaurus; for relatedness, a word association data set is used. The resulting word vectors show improvements over word2vec in both word similarity tasks and downstream applications such as document classification. However, improvements in word similarity can also be achieved using lower-level information such as lemmas and part of speech tags to improve syntactical disambiguation and reduce vocabulary size. Such simple, automatic augmentation can improve performance in word similarity tasks, especially for verbs, see also Kuznetsov and Gurevych [58]. Similar approaches can be used to encode semantic information using constraints over monolingual and cross-lingual data sets that require word vectors to be more or less similar [85].

A more principled approach to the evaluation of word representations should try to understand what information is and is not encoded by various representations. Such understanding can provide guidance towards intentional applications of representations in downstream tasks. The need for such approaches to evaluation is emphasized by revelations that word2vec spaces are unstable and might encode different kinds of information given small changes in the training corpus [3]. And, maybe even more importantly, that word representations can encode societal biases which might affect the downstream applications relying on them [14, 19]. Evaluation of word representations is not a settled topic, but an active research field as indicated by the continuously increasing number of publications in dedicated workshops like *VecEval* and *Repl4NLP*. This section is not meant to be an exhaustive analysis of all evaluation methods, but rather to emphasize the importance of understanding the kind of information encoded in word representations, how it can affect downstream tasks and give the reader an overview of the current evaluation methods and their limitations. To readers interested in following up on evaluation of word representations, I recommend the survey of Schnabel et al. [105], the multi-method evaluation framework created by Faruqui and Dyer [42], and proceedings from recent editions of *VecEval* and *Repl4NLP* workshops mentioned above.

3.4 Conclusion

In this chapter, I built on the knowledge of language models introduced in Chapter 2 and presented a number of methods for discrete and continuous word representations using language models. I discussed a number of problems affecting derivation of word representations such as computational complexity and requirements on the amount of data. I emphasized evaluation and understanding of information encoded in word representations and discussed the benefits and drawbacks of various
evaluation methods. Finally, I argued that a principled methodology is required to understand what kinds of information are encoded in word representations and the limitations of each method in order to support informed and intentional use of word representations in downstream Natural Language Processing tasks.
Chapter 4

Contributions and Future Work

In Chapter 3, I introduced various kinds of word representations, presented challenges and open questions concerning them, and argued for the need to understand what information word representations encode in order to intentionally use them in downstream Natural Language Processing applications.

In this chapter, I discuss my contributions to the open questions that I identified regarding word representations, and outline future work. In Section 4.1, I present work that quantifies the amount of morphosyntactic information encoded in Brown clusters and briefly outline an intentional application of Brown clusters in Neural Machine Translation. Following, in Section 4.2, I present insights that allow for faster computation of Brown clusters that do not require new implementations or specialized hardware. Finally, in Section 4.3, I present my work on word representations for abbreviation disambiguation using unsupervised learning and without using hand-designed language-specific features.

4.1 Quantifying the morphosyntactic content of Brown clusters

In Quantifying the morphosyntactic content of Brown Clusters [32] (included in Chapter 5), we set to answer some questions presented in Section 3.3, more specifically understanding what kind of information is encoded in word representations. As mentioned in the previous chapter, word clusters have been used as word representations in numerous downstream Natural Language Processing (NLP) tasks such as Part of Speech tagging [39, 94, 116], Named Entity Recognition [39, 64], Dependency Parsing [8, 57], Chinese word segmentation [64], and identification of Named Chemical Entities [116]. Unfortunately, despite their importance in these tasks, it is not known what kind of information they contain. Because of this, it is not known how and which downstream NLP tasks could benefit from using Brown clusters.

We empirically measure the amount of morphosyntactic information contained in Brown clusters constructed over corpora from several languages belonging to different language families: Germanic (English), Slavic (Czech), and Latin (French).
We measure the separation of the vocabulary by Part of Speech as a proxy for understanding how much morphosyntactic information the clusters contain. Part of Speech tags, or Grammatic Equivalence Classes [70], indicate the syntactic role of words in a language. Depending on tag set design, Part of Speech tags can also include morphological information (e.g. tenses or number). Therefore, measuring how well Brown clusters match Part of Speech tags allows us to quantify the amount of morphosyntactic information they encode. Clusters containing syntactic role information are good candidates to represent words in downstream tasks that rely on such information, e.g. Named Entity Recognition. Unlike Christodoulopoulos et al. [25], or more recently Cardenas et al. [21], our goal is not unsupervised Part of Speech labeling, but understanding the word clusters themselves.

We used the Universal Dependencies corpus [59] in order to access data in three languages from different language families: English (Germanic), French (Latin), and Czech (Slavic), with manually annotated sentences, tokens, and Parts of Speech (PoS). This experimental design has several benefits. By using data that has manual annotations for sentences and tokens, we eliminate pre-processing noise. The variety of languages with different amounts of morphological inflection allows us to understand how the clustering algorithms behave and generalize our conclusions.

In all our experiments, we ran both the Exchange and Brown clustering algorithms. We have shown that once low-frequency words are eliminated, even when the number of clusters formed exactly matches that of PoS tags, 67.74% – 78.60% of words in the vocabulary (depending on the language) are properly classified according to their Part of Speech tag. Even when using automated PoS tagging on the EuroParl corpus, the percentage remains high, although decreased. As the number of clusters is increased to 100, vocabulary separation accuracy increases to 82.4% – 85.6% (depending on the language). Furthermore, improvement in vocabulary separation is highly correlated with an increase in Average Mutual Information which suggests that algorithmic improvements to the Exchange or Brown clustering algorithms could lead to improved vocabulary separation by PoS.

The results explain why Brown clusters have been so successful in downstream tasks that rely on syntactic information such as Part of Speech tagging in clean and noisy text [39, 94, 116], Dependency Parsing [8, 57], and unsupervised PoS induction [21, 25]. Furthermore, our research provides a basis for choosing word representations in downstream NLP tasks. Instead of experimentation using multiple kinds of word representations, or looking up word representation choices for similar tasks in NLP literature, one can choose to use Brown clusters as word representations for those downstream tasks that require syntactical information.

Future work

Brown clusters for low resource Neural Machine Translation (NMT)

Machine Translation using Neural Networks (also called Neural Machine Translation, or NMT) [115] has shown improved performance over Statistical Machine
4.2. ACCELERATING COMPUTATION OF BROWN CLUSTERS

Translation (SMT) [16]. However, neural networks are data-hungry and, in the absence of large parallel corpora, NMT cannot surpass the performance of SMT models [112]. The small amount of aligned parallel corpora affects deep end-to-end NMT systems which need lots of data as the network must first learn the morphology, syntax, and meaning of each language, and then learn how to map utterances from one to the other [112, 115]. This is a limiting factor for Machine Translation systems and leads to increased costs since creating large parallel corpora is a manual task that requires humans to create large corpora of text translated into multiple languages. In the case of rare languages, it can prevent the creation and deployment of effective machine translation systems. Using subword information through Byte Pair Encoding addresses some issues, but low-resource machine translation is still an active research topic [107].

Previous research into low resource SMT has shown that inclusion of morphosyntactic information can improve translation quality [87]. Qu et al. [98] have shown that Brown clusters can learn effective representations even from small monolingual corpora, while my research has shown that Brown clusters encode syntactic information [32]. Thus, Brown clusters are prime candidates for providing additional information to deep NMT systems. I propose to study Brown clusters as supplementary, non-trainable, input to deep NMT systems, alongside tokens from small parallel corpora. I hypothesize that Brown clusters on the source language, target, or both can improve translation performance even when using small parallel corpora. My expectation is that Brown clusters provide a supplementary signal of syntactic role of words that NMT systems can directly learn to use, and thus eliminate the need for the much more complex and slower neural network to learn the same information. For evaluation, I can use already available large parallel corpora (e.g. English-German) and test by incrementally reducing the amount of parallel information available for training.

4.2 Accelerating computation of Brown clusters

As explained in Section 3.1, Brown clusters are slow to compute since their computation cannot be easily parallelized. Average Mutual Information (AMI), the global optimization function that guides both the Brown clustering algorithm and the Exchange algorithm, is affected by any change to the contents of clusters, for example after every merge of two clusters in Brown clustering, or a word moving to another cluster in the case of Exchange. To explain the problem, we will use an example. Consider two words $w_a$ and $w_b$ currently assigned to clusters $c_x$ and $c_y$, respectively. When using Exchange for clustering, we want to consider moving $w_a$ and $w_b$ to every other possible cluster and compute the resulting AMI. We can parallelize this analysis by assigning one thread to each word. If we are in a situation where both $w_a$ and $w_b$ should move from their current clusters $c_x$ and $c_y$ to some other clusters, say $c_z$ and $c_t$, then moving both in parallel could lead to a clustering with lower AMI. Average Mutual Information is affected by any operation on the clustering, if we move $w_a$
first, then we may invalidate the analysis for \( w_b \) and the other way around. Thus, the parallel algorithm becomes sequential since, after every move, we must invalidate all other analyses, effectively only allowing for parallelism within the analysis performed for each word. The same problem arises in the Brown clustering algorithm where we cannot perform multiple merges in parallel since that may not result in the least expensive merge at every step.

The ineffectiveness of the Brown and Exchange clustering algorithms to take advantage of highly parallel processors has resulted in limitations on corpus length and vocabulary size keeping the number of clusters between hundreds and several thousands, even with today’s computing power [17, 38, 39, 53, 72]. The problem is compounded for Exchange by suggestions in the literature that the algorithm should be run for \( 15 - 21 \) iterations [72, 120]. Yet another roadblock for algorithm performance is the lack of fast, modern implementations of both algorithms. \textit{wcluster} [64] is the only widely available implementation of Brown clustering and is still relatively fast on modern CPUs, despite its age. Unfortunately, it only contains the Brown clustering algorithm and thus does not support comparisons with Exchange or interconnected runs of Exchange and Brown as suggested by Brown et al. [17]. While Brown clustering is less data-hungry than word representation methods using vector spaces [98], for Brown clusters to become feasible in downstream applications, faster computation methods are still needed.

In \textit{Accelerated High-Quality Mutual-Information Based Hierarchical Clustering} [31] (included in Chapter 6), we provide an open implementation of both Brown and Exchange clustering. It is the only open code base that includes implementations of both algorithms and allows for output from one algorithm to be used in the other.

We propose running Exchange [53] in order to create a flat clustering containing \( k \) clusters, and then construct a hierarchy over the clusters using Brown [17]. Our experiments show this hybrid approach to be up to twice as fast as computing clusters using Brown clustering alone. Besides the 2x speed improvement, our hybrid approach achieves higher Average Mutual Information (AMI) meaning that the underlying class-based bi-gram language model is better able to predict the training corpus. Downstream experiments on Named Entity Recognition show that clusters computed using the hybrid approach match the performance of those derived exclusively using Brown clustering. This is independent of the \( k \) hyper-parameter determining the number of clusters. We also show that a stochastic version of the Exchange algorithm where a small percentage \( p \) of moves are performed at random rather than in order to increase AMI can result in higher final AMI at the same cost as a regular run of Exchange. We leverage our unique codebase containing implementations of both Brown and Exchange to empirically study the two algorithms’ behavior on languages from three different families: Germanic (English), Slavic (Czech), and Latin (French).

Moreover, based on empirical evaluation, we provide practical guidance for speeding up Exchange clustering. By tracking Average Mutual Information and swaps through iterations using different values for the number of clusters hyper-parameter \( k \), over the three different languages, we show that most of the Average Mutual In-
4.3. UNSUPERVISED ABBREVIATION DISAMBIGUATION

formation (AMI) improvement in the Exchange algorithm (over 98%) is achieved during the first 3 iterations of the algorithm. Furthermore, after the first 3 iterations, less than 10% of the vocabulary still participates in swaps. Therefore, stopping Exchange after just 3 iterations can provide a 5 to 7 times speedup in computation time compared to the 15 to 21 iterations common in literature [72, 120]. Taking it to the extreme, we measure AMI with Exchange after a single iteration and find that at low values of clusters $k$, single-iteration Exchange provides 50.5% of the final AMI from 10-iteration runs of Exchange, but with higher values of the hyper-parameter $k > 600$, the percentage increases to 78.5% – 85.5%. This finding suggests that for large numbers of clusters (high values of $k$), a single iteration of Exchange might be a feasible trade-off between AMI and run time.

Our proposals for speeding up the computation of Brown clusters by using a hybrid Exchange-Brown run, and reducing the number of iterations for Exchange are easily applicable as they do not require massively parallel hardware and, in fact, do not even require modifications to the clustering algorithms themselves.

Future work

The insights gained while performing this work suggest that more speedups can be achieved. More specifically, the small number of words involved in swaps with the Exchange algorithm suggests that it might be possible to partially parallelize the algorithm by running swap analyses for multiple words in parallel and only discard analyses when a swap has taken place, maybe even select which word to move based on the highest increase in AMI. Another speedup might be achievable by ignoring the effect of swaps on Average Mutual Information and avoiding re-computation. This would result in a parallel version of Exchange that behaves differently from the single-threaded version. However, that might not be a problem if AMI is improved as the computation progresses and the parallel version is faster than the sequential-move version of the algorithm implemented in our code (based on Kneser and Ney [53]). However, since the final clusters and AMI values are language-specific, it is impossible to perform theoretical analyses on such a modification of the algorithm. Therefore careful empirical analysis is required, on multiple languages where different orders for move operations are considered of corpora distinct styles and predictability.

4.3 Unsupervised Abbreviation Disambiguation using word vector spaces

Abbreviation disambiguation is the task of determining the correct meaning (referred to as long-form) for an abbreviation (termed short-form) given an example of using the abbreviation in context. If the abbreviation has only one meaning, the task can be accomplished via a dictionary lookup. However, if multiple concepts are denoted by the same abbreviation, then the context around an abbreviation becomes crucial for resolving the intended meaning (long-form). Humans generally have problems solv-
CHAPTER 4. CONTRIBUTIONS AND FUTURE WORK

ing the abbreviation disambiguation task mostly due to their limited domain knowledge.

Existing abbreviation disambiguation methods support a rather small number of abbreviations, rely on intricate ways to represent context, and are supervised systems that require domain experts to label data from which to train disambiguation systems [63, 79, 80, 123, 124]. The small number of abbreviations supported is a bi-product of the small data sets available for learning. Some systems [124] even use a classifier per abbreviation and support less than 100 distinct abbreviations. Such architectures are not suited for industrial deployments where systems are expected to support hundreds or thousands of abbreviations. Context representation is problematic as most systems rely on hand-designed, language-dependent, representations of context that use word co-occurrence, the number of uppercase letters, prefixes, suffixes, and other character-level features [79, 80, 123]. In Unsupervised Abbreviation Disambiguation - Contextual disambiguation using word embeddings [33] (included in Chapter 7), I propose UAD, a method that addresses several of the drawbacks mentioned above. It is unsupervised, requires no hand-designed language-specific features, and supports disambiguation of thousands of abbreviations in the same model. UAD supports thousands of abbreviations by creating word representations for context and for abbreviation meanings. Besides support for thousands of abbreviations, the reliance of word representations allows for inspection, understanding, and correction of challenging long-forms without requiring manually annotated data. We call this process pre-evaluation and discuss it in detail in the next section.

An important aspect of Unsupervised Abbreviation Disambiguation (UAD) is its unsupervised nature which means that we do not need manually labeled corpora of abbreviations. First, we use a rule-based approach on large corpora in order to extract sentences containing uses of abbreviations, but where the intended meaning is known and establish candidate meanings (lists of potential long-forms associated with each short-form). These can be sentences containing abbreviation definitions (such as the first sentence in this paragraph), or followup sentences where the meaning is known due to a preceding definition (like the second sentence in this paragraph). While this method might have a low recall, its high precision combined with the large corpus results in a reliable data set of unambiguous uses of abbreviations. Following, we devise another rule-based method to normalize variations in abbreviation long-forms. This step is required in order to eliminate common inconsistencies in the spelling of long-forms such as plurality, hyphenation, capitalization, differences between British and American spelling, or simply typographical errors. As part of the normalization step, we identify unambiguous and ambiguous abbreviations, for example, in all of English Wikipedia the abbreviation NASA is used exclusively as an acronym for National Aeronautics and Space Administration. Unambiguous abbreviations are of no

---

1An unambiguous use of an abbreviation is a sentence where we know the intended meaning of an abbreviation. An ambiguous abbreviation is one that has multiple long-form candidates for expansion. Subsequently, an unambiguous use of an ambiguous abbreviation is a sentence containing an abbreviation that has multiple long-form candidates for expansion, but where we know the correct expansion intended by the author of the sentence.
interest for UAD as their expansion is, in fact, just a dictionary lookup. A real-world deployment of an abbreviation expansion system should handle both ambiguous and unambiguous abbreviations. We discuss this in the following subsection.

The second unsupervised aspect is the vector space construction itself. Following identification of normalized long-forms, we formulate the abbreviation disambiguation problem as the task of predicting the missing token in a sequence (following the discussion in Section 2.2). We replace the occurrence of an abbreviation short-form and any following parenthesis and tokens constituting the long-form with a single token that identifies the abbreviation and its intended meaning. With this corpus we train a word2vec vector space. Since word2vec computes word vector representations by training a language model to predict the missing word (see Section 3.2), the resulting vector space contains one vector for every single word in the vocabulary and one for every identified long-form. All word vectors and those assigned to tokens that denote abbreviation meanings are already placed in relation to each other based on their usage in the corpus. At disambiguation time, we construct a vector to represent the surrounding context by summing up vectors for all surrounding words and compare the context vector with each candidate long-form vector. The formulation of the disambiguation problem as a word prediction task to be solved using word2vec’s language model allows us to avoid (1) hand-designed language-specific features used in previous literature [79, 80, 123] and (2) hand-designed weighing functions such as those used by competitor vector-based disambiguation methods [22, 63].

Few existing abbreviation disambiguation methods use word representations to represent the context around a short-form. Notably, Charbonnier and Wartena [22], Li et al. [63] take a step in this direction by using word vector representations combined with weighing functions inspired by the information retrieval literature. UAD does not use any weighing functions. Instead, it builds vector-based word representations. This not only eliminates the need for weighing functions, but also makes UAD compatible out of the box with new languages, and due to its unsupervised nature, easy to use on new domains.

In experimental evaluation over two large data sets consisting of all English Wikipedia and the commercially friendly subset of the open-access subset of PubMed, UAD achieves disambiguation accuracy of 94.28% on Wikipedia and 77.62% on PubMed significantly outperforming both a frequency-based baseline and the two state-of-the-art competitors that use word vectors [22, 63]. UAD’s performance shows that carefully designed word representations in Natural Language Processing systems can help eliminate hand-designed features, weighing strategies, and avoid the need to train and deploy one model per abbreviation, thus making systems scalable to industrial needs, as well as and language-independent.

**Integrating UAD into production systems and analyzing its performance**

Evaluation of abbreviation disambiguation systems is usually a process that involves training a method, and then evaluating the resulting language model on manually labeled data. This is a costly process both in terms of the time required to train
and run evaluations, but also due to the cost involved in manually labeling data. A unique design aspect of UAD’s construction of the word and long-form representation vector space is that it allows for performance analysis through a process we call pre-evaluation. In Abbreviation Explorer - an interactive system for pre-evaluation of Unsupervised Abbreviation Disambiguation [29] (included in Chapter 9), I present an interactive system that allows users to inspect pairs of long-forms that are challenging for UAD to distinguish between. Such an analysis is made possible by UAD’s construction of word representations. In UAD, disambiguation of abbreviations involves comparing the vector representing context in the sentence with each candidate long-form. The comparison is based on the cosine distance. Since all long-forms belonging to a given short-form are represented in the same vector space, it is also possible to compute cosine distances between the long-forms themselves. Thus, if two long-forms belonging to the same short-form are in close cosine distance to each other, then it is likely that if a sentence context is close in cosine distance to one of them, it will also be close to the other. Subsequently, which long-form is emitted by UAD as disambiguation will depend on small changes to the context vector making the disambiguation more error prone. Such problematic pairs of long-forms can appear when not enough data exists for long-forms in similar domains, but more often it appears for non-trivial variations in long-form spelling. After the automatic identification of problematic long-forms, users can then choose to delete one of the long-forms or, in cases where they denote the same concept, rewrite one long-form to the other. Our experiments show that on the Wikipedia data set used in our UAD paper [33] discussed above, we can improve accuracy by up to 1.8 percentage points without even requiring re-computation of the vector space.

In systems that use classifiers and hand-designed language-specific features [79, 80, 123], such an analysis is impossible as the disambiguation models are opaque and the only way to understand performance is through evaluation on manually annotated data.

Abbreviation disambiguation systems usually work exclusively with ambiguous abbreviations (those that have more than one possible expansion). However, one important aspect for industrial deployments of abbreviation expansion systems is the integration of abbreviation disambiguation with other data sources for expansion of abbreviations that might not involve disambiguation. For example, unambiguous abbreviations should also be expanded and document-specific definitions should be taken into account when expanding abbreviations. In Abbreviation Expander – A web-based system for easy reading of technical documents [30] (included in Chapter 8), I present an architecture that allows for deployment of UAD (or any other abbreviation disambiguation system) together with dictionaries containing single-meaning abbreviations, and with author-defined abbreviations. The architecture allows for multiple candidate expansions from different sources to be considered for each usage of an abbreviation. Thus, for unambiguous abbreviations, the meaning annotated (i.e., long-form) is the one identified in the dictionary of single-meaning abbreviations. Another component identifies abbreviation definitions in the document and resolves short-forms to the long-form used by the author using the assumption
4.3. UNSUPERVISED ABBREVIATION DISAMBIGUATION

Figure 4.1: System architecture: input text is split and tokenized, defined abbreviations are extracted and expanded (Pattern Annotator), a dictionary resolves unambiguous cases (Dictionary Annotator), ambiguous cases are expanded using context (Vector Space Disambiguator); possible conflicts are resolved in the Expansion Combiner.

of one meaning per document. Lastly, UAD is deployed to disambiguate in context. Following the annotators, a component combines the answers using priority rules to create consistent disambiguations. For more details, see Figure 4.1 reproduced from Chapter 8. The architecture provides several contributions. First, it allows for abbreviation expansion and disambiguation to be consistent both with author intent (by prioritizing abbreviation definitions in the document), and with itself (by normalizing all UAD expansions to a single meaning in the respective document). Since UAD disambiguates abbreviations one a sentence-by-sentence basis, it is possible for different sentences using the same abbreviation in the same document to be disambiguated to different meanings. Therefore, a component in the system takes a simple majority vote of the disambiguated meaning of each abbreviation and enforces the most popular disambiguation on all sentences in the document, thus ensuring consistency. Secondly, the combining component itself can be used to identify documents where inconsistencies exist in the disambiguations from UAD, such documents can combined with manual labeling in order to create data sets of sentences where disambiguation is difficult, thus providing more training data. Finally, the system is transparent to the user. Every abbreviation expansion is annotated with a reason for its expansion whether based on dictionary, the definition in the document, or contextual disambiguation performed by UAD.

Future Work

Improved context representation for abbreviation disambiguation

My research on abbreviation disambiguation shows that careful construction of word representations for automatically extracted long-forms can be used to build state-of-the-art abbreviation disambiguation systems. In fact, I show that relying on word representations constructed by a language model is more effective for abbreviation
disambiguation than language-specific hand-designed features or weighing strategies for constructing fingerprints from the context [33]. However, my experiments show that both UAD, the method I propose, and other vector-based competitors suffer from context pollution by irrelevant words that add noise (e.g. stop words or tokens contained in sentence clauses that do not add information about the abbreviation meaning).

Strategies adopted so far are hand-designed: removing stop-words from context [22, 33, 63] and hand-designed term weights [22, 63] ignore sentence structure, require manual work when using different languages, and tend to ignore properties of natural language such as long-distance relationships (such as the ones we discussed in Chapter 2), morphology or sentence structure. To achieve even better abbreviation disambiguation, we must carefully consider what constitutes the context relevant to each abbreviation use. I would like to study two filtering strategies: one based on Parts of Speech, and the other based on relations in the dependency tree of each sentence. Nouns, adjectives, and verbs tend to carry most information in sentences. They are therefore prime candidates for constructing a context based on Parts of Speech. Dependency Relations (discussed in Chapter 3) allow this idea to be taken a step further by focusing only on tokens in direct relation to the abbreviation use. Context filters based on Parts of Speech and dependency relations do not require manually constructed glossaries or weighing functions. However, they do require some language-specific, manually specified filtering rules. On the other hand, deep neural network representations based on language models are able to capture long-distance relationships. ELMo [97] relies on a linear combination of bi-directional language models that allows it to learn by itself what relation types and context lengths are relevant. I expect that maintaining the language model prediction formulation from my proposed method, but using deep language model representations such as ELMo will improve disambiguation performance while still allowing for learning word representations from unstructured text that does not contain abbreviations.
Part II

Publications
Chapter 5

Quantifying the morphosyntactic content of Brown Clusters

Abstract

Brown and Exchange word clusters have long been successfully used as word representations in Natural Language Processing (NLP) systems. Their success has been attributed to their seeming ability to represent both semantic and syntactic information. Using corpora representing several language families, we test the hypothesis that Brown and Exchange word clusters are highly effective at encoding morphosyntactic information. Our experiments show that word clusters are highly capable of distinguishing Parts of Speech. We show that increases in Average Mutual Information, the clustering algorithms’ optimization goal, are highly correlated with improvements in encoding of morphosyntactic information. Our results provide empirical evidence that downstream NLP systems addressing tasks dependent on morphosyntactic information can benefit from word cluster features.

5.1 Introduction

Distributionally generated word classes (often referred to as word clusters) are hard clusters, containing all word types observed in a corpus, allocated to clusters based on contextual information observed in the corpus. They have found wide use in Natural Language Processing (NLP) systems as an alternative to word embeddings such as word2vec [75]. Word clusters differentiate themselves from word embeddings by requiring estimation of many fewer parameters, and by their ability to derive qualitative representations from smaller corpora [8, 98].

Brown Clusters [17] are a well-known approach based on hard, hierarchical, distributionally derived groups of word types observed in a corpus of unstructured text, with Average Mutual Information (AMI) as the optimization goal. Exchange Clusters are an alternative approach obtained by applying the Exchange Algorithm [53] to the same optimization goal. Unlike Brown, Exchange outputs a flat clustering, with
no hierarchy [72]. When only the bottom of the hierarchy is used, like in this paper, Exchange and Brown clusters are interchangeable.

Both Brown and Exchange clusters have been used as word representations for various Natural Language Processing tasks such as Part of Speech tagging in clean and noisy text [39, 94, 116], dependency parsing [8, 57], Chinese Word Segmentation [64], and Named Entity Recognition [39, 64, 116]. Word clusters distinguish themselves from word embedding models by their ability to learn from little data [8, 98]; for example, in cases like [8], word clusters outperform other kinds of representations, including word embeddings. In the literature, it is often observed that word clusters seem to encode a considerable amount of morphosyntactic and semantic knowledge [17, 39]. However, it has not yet been studied to which extent such knowledge is encoded, as previous work on Brown and Exchange clusters focuses mostly on algorithmic improvements and on applications to different NLP tasks.

In this work, we present a principled study of the morphosyntactic information encoded in flat word clusters induced exclusively from class-based language models via Brown Clustering and Exchange algorithm. In particular, we focus on how well these approaches derive clusters that represent Parts of Speech as a measure of the morphosyntactic information encoded.

We find that Brown and Exchange clusters are highly effective at representing morphosyntactic information, even when hyper-parameters are set such that they match only the number of Parts of Speech, thereby grouping into relatively few word clusters only. Our results provide empirical evidence for the observed performance gains when including Brown and Exchange word clusters as features in NLP systems that rely on morphosyntactic information.

Furthermore, we find that there is a strong correlation between the optimization goal of Brown clustering and the Exchange Algorithm (i.e., Average Mutual Information), and performance at Parts of Speech separation, which again confirms the appropriateness of choosing AMI in word clustering for morphosyntactic information.

5.2 Background

Class-based language models address the problem of brittleness in classic n-gram language models by trading precision for performance stability over different text styles [17].

Brown Clustering [17] and Exchange [53] are greedy algorithms that construct word classes by optimizing for higher Average Mutual Information (AMI). Maximizing Average Mutual Information is a proxy for maximizing the log-likelihood of the underlying class-based language model on the given corpus [72]. Despite their age, most research on Brown or Exchange clusters has so far followed two major directions: algorithm improvements and applications in Natural Language Processing. In contrast little focus has been placed on understanding and evaluating the information content of the clusters.
In the direction of algorithm improvements, work has been done on the effect of greedy merge choices in Brown Clustering [27, 38] and extension of AMI to n-grams [72]. Model relaxations, particularly to Exchange, aim to improve computational performance by reducing the effect of words swapping clusters [36, 120].

As mentioned earlier, both Brown and Exchange clusters have seen many applications in Natural Language Processing (NLP) systems: PoS tagging [39, 94, 116], dependency parsing [8, 57], Chinese Word Segmentation [64], and Named Entity Recognition [39, 64, 116]. Most of this work, like [116] uses the word clusters as sources of features which are combined with hand-designed ones. While word clusters derived using Exchange and Brown clustering have found wide use in NLP systems, their use has been based on the assumption that they encode morphosyntactic and semantic information rather than a principled use.

In relation to Parts of Speech, early on Martin et al. [72] concluded that initializing Exchange with PoS-homogeneous clusters has no effect on final clustering AMI, but that it does help accelerate convergence. More recently, Christodoulopoulos et al. [25] found that Brown clusters match the performance of more sophisticated clustering methods, despite their simple algorithmic construction. The study focused on using word clustering algorithms as sources of prototypical information to prototype-driven learning models for classification. In this paper, we study the amount of morphosyntactic information encoded in Brown and Exchange word clusters with the goal of providing empirical results for a principled use of such clusters in downstream tasks.

5.3 Metric selection

In order to determine the amount of morphosyntactic information encoded in Brown and Exchange word clusters, we measure their ability to separate word types by their Parts of Speech. For this, we require cluster quality measures. Brown and Exchange clusters do not exist in a metric space; therefore unsupervised cluster quality measures relying on distances between points or clusters, such as the Silhouette coefficient [104], cannot be used. Instead, we focus on two quality measures that compare clusters with a ground truth partitioning. We work under the hypothesis that Brown and Exchange clusters represent parts of speech and thus, we consider parts of speech as the ground-truth partitioning of the data. This makes it possible to use cluster quality measures that require as input an existing ground-truth partitioning. We use PoS tags resulting from manual or automatic annotation. We evaluate using a widespread and easy to interpret measure based on overlap (purity), and an information theoretical measure (Adjusted Mutual Information).

Clustering Purity

Cluster purity measures how many points in a clustering (in our case words) have been assigned to a cluster whose predominant label they share (e.g. adjectives clustered with other adjectives, nouns with other nouns etc). Intuitively, it measures the
percentage of points properly classified (via their cluster membership). Formally, cluster purity is defined as:

\[
purity(C_i) = \frac{1}{|C_i|} \max_{l=1}^{L} |\text{label}(C_i, l)| \quad (5.1)
\]

\[
purity(C) = \sum_{i=1}^{k} \frac{|C_i|}{|V|} \cdot purity(C_i) \quad (5.2)
\]

\[
= \frac{1}{|V|} \sum_{i=1}^{k} \frac{|L|}{\max_{l=1}^{L} |\text{label}(C_i, l)|} \quad (5.3)
\]

Where the function \( \text{label}(C_i, l) \) provides the number of elements from \( C_i \) with label \( l \) and \( L \) is the set of labels. Purity reaches a value of 1 when the clustering is identical to the ground-truth partitioning, or each point is allocated to its own cluster (i.e. \( k = |V| \)). When \( k = 1 \), purity is equal to the fraction of points labeled with the most popular label, and thus provides a baseline. In our case, that is equal to the percentage of vocabulary allocated to the most popular PoS class, usually nouns. For values of \( k \in (1, |V|) \), it varies depending on cluster quality. If \( k > |L| \), purity can take a value of 1 if each cluster is a subset of a ground partition. Thus, purity is expected to increase as \( k \) grows higher than \( |L| \).

In our experiments, purity measures the percentage of vocabulary that is labeled correctly. In other words, purity does not depend on word frequency. Thus, it is not an approximation of PoS tagging accuracy, like the M-1 measure used by Bansal et al. [8]. Since we focus on morphosyntactic information encoded in word clusters, we do not want a measure that takes into account word frequency in the given corpus (i.e., one that is a good approximation of PoS tagging performance), but one that focuses exclusively on the clusters and their content.

**Adjusted Mutual Information**

Since the vocabulary size is fixed, as the number of clusters \( k \) increases, purity can increase even if cluster membership is randomly assigned, as it is easier for smaller clusters to randomly achieve label agreement. **Adjusted Mutual Information (AdjMI)** [121], not to be confused with AMI (Brown and Exchange’s optimization goal), measures the amount of information shared by the ground truth partitioning \( U \) and a clustering \( C \). In our evaluation that corresponds to the amount of information shared by the PoS ground-truth partitioning and the clustering resulting from Brown or Exchange. AdjMI corrects for the mutual information expected to exist between the ground truth partitioning \( U \) and a random clustering. Formally, it is defined as:

\[
\text{AdjMI}(U, C) = \frac{MI(U, C) - E\{MI(U, C)\}}{\text{avg}\{H(U), H(C)\} - E\{MI(U, C)\}} \quad (5.4)
\]
5.4. EXPERIMENTS

Data and preprocessing

We use manually annotated data from Universal Dependencies (UD) [59] for English, French and Czech\(^1\). We chose the group of languages so that it represents different language families. Our choice of languages is based on the amount of manually labeled data, and the presence of each language in the larger, not annotated, EuroParl corpus. We append the manual or automatic PoS tags and convert text to lowercase. Therefore, a sentence such as “Words have meaning,” is transformed into “words\_NOUN have\_VERB meaning\_NOUN \_PUNCT”. Both word clustering algorithms studied in this paper are insensitive to the appended PoS tags as they operate at word and not character level. The appended tags allow us to evaluate the quality of word clusters using the measures described in the previous section. We replace all numbers, dates, times, URLs and emails with placeholders in order to reduce vocabulary size. Universal Dependencies is the largest manually annotated corpus we have access to. For experiments on larger corpora, we use the unlabeled EuroParl corpus [54]. More specifically, the English-French and English-Czech pairs. Since manually annotated PoS tags are not available for EuroParl, we append automati-

---
\(^1\)We use data from release 2.2 of Universal Dependencies.
cally assigned PoS tags, obtained by using UDPipe [113] pre-trained on manually annotated corpora from Universal Dependencies.

We use flat clusters from the Exchange clustering algorithm for all experiments reported in this section as they outperform the flat clustering resulting from Brown in terms of Average Mutual Information (their optimization goal), Adjusted Mutual Information (AdjMI) and cluster purity. All observations in the following section also apply to the flat clusters resulting from Brown. For interested readers, we include all experiments with Brown Clustering as supplementary material. The fact that Exchange outperforms Brown Clustering in terms of AMI is well-understood [17], but its effect on cluster content is not.

**Morphosyntactic content in Exchange and Brown clusters**

Using Exchange, we induce flat clusterings with $k$ in the range 18 to 800. We start with $k = 18$ as it matches the observed number of distinct PoS tags in the Universal Dependencies corpora (17 distinct tags and one catch-all tag). When setting the hyper-parameter $k$ to be higher than 18, if Exchange separates clusters by Parts of Speech, then the expectation is that clusters are subsets of words sharing the same PoS tags, and that purity for such clusterings will be high. In Figure 5.1a, we show purity measured on the aforementioned clusters. We can see that, even when the number of clusters is equal to that of PoS tags ($k = 18$), between 55% and 62% of the vocabulary is properly separated. Purity increases as $k$ increases towards 100. At $k = 500$ and $k = 800$, between 64% and 70% of the vocabulary is grouped based on PoS, not that much more than at $k = 100$.

Increasing the number of clusters $k$ to high values is not guaranteed to improve purity, for any of the languages studied. This is contrary to the expectation that purity increases when $k > 18$. This indicates that Exchange and Brown do not exclusively optimize for Part of Speech separation. We believe the clustering algorithms might be striking a balance between encoding semantic and morphosyntactic information, since at higher values of $k$ we usually see more clusters with a coherent semantic theme such as names of geographic locations, names of men, names of women, nouns determining times, similar to the clusters observed in previous literature [17]. For example, when using $k = 18$ in English, the token “cat\_NOUN” appears in the same cluster as the plural version “cats\_NOUN”. At $k = 800$, the clusters distinguish between the two tokens and “cat\_NOUN” is placed together with a number of nouns in the singular such as “budget\_NOUN, computer\_NOUN, pet\_NOUN, wheel\_NOUN”, while “cats\_NOUN” is placed in a cluster of mostly pluralized nouns like “children\_NOUN, rooms\_NOUN, dogs\_NOUN, families\_NOUN”.

Adjusted Mutual Information (AdjMI) for the same clusterings, Figure 5.1b, shows a considerable decrease as the hyper-parameter $k$ increases, especially at high values of 500 and 800. This is in line with the expected punishment due to the effect attributed to randomness (see the term for expected value of Mutual Information in Equation (5.4)). At values of $k$ closer to the number of PoS tags in the data, AdjMI varies little from one clustering to the other. More interestingly, the relative order
5.4. EXPERIMENTS

of separation performance between the languages studied is maintained going from purity to AdjMI, suggesting that no measure-specific effects are at play.

By studying the frequency of incorrectly classified words types (i.e., of those whose PoS tag does not match the most popular one in their cluster), we find that most (about 85%) occur less than 5 times in the corpora. Such few observations likely do not provide enough information for Brown or Exchange to properly place those words. Therefore, from the already computed clusterings, we remove words with a frequency less than 5 and recalculate the two quality measures.

In Figures 5.1c and 5.1d, we can see that both purity and AdjMI improve considerably. Even in the most difficult case ($k = 18$), where the number of clusters matches that of distinct PoS tags, between 68% and 78% of words are properly placed, an increase of 21% – 28% compared to the values in Figure 5.1a. For AdjMI, the scores more than double. These results show that even for small corpora, a large amount of morphosyntactic information can be encoded, completely unsupervised, using the Exchange clustering algorithm. (The same behavior can be observed for clusters derived using the Brown Clustering algorithm, see supplementary material.) It also shows that, for low frequency terms, there is not enough contextual information for a proper clustering.

Figure 5.1: Cluster agreement with manual labels from UD.
One disadvantage of thresholding by frequency is that, due to the Zipfian distribution of word frequencies in natural language, only a fraction of the original vocabulary remains after filtering out words with a frequency less than 5: English (8,143 words – 24.01%), French (9,020 words – 17.45%), Czech (37,026 words – 22.51%). In order to benefit from more reliable word usage estimates, it is necessary to perform the same experiment on larger corpora. Unfortunately, bigger manually annotated data sets do not exist. We therefore turn to automatic PoS tagging.

We use UDPipe [113] with models pretrained on the Universal Dependencies corpora to automatically tag text from the EuroParl multi-language corpus containing transcriptions of European Parliament proceedings [54]. Automated annotations introduce labeling noise that should lead to a decrease in separation performance. Despite this, we expect to still be able to observe good PoS separation.

After filtering the EuroParl corpora, the size of remaining vocabulary is considerably larger: English (60,373 words – 37.80%), French (78,822 words – 38.60%), Czech (62,512 words – 35.19%). In Figure 5.2a, we can see that there is a drop in performance that varies with language, but when looking at purity, even in the worst performing clustering (French at $k = 50$), 60% of the vocabulary is still properly separated according to Parts of Speech. A drop in performance can also be observed for AdjMI in Figure 5.2b, with the value dropping for all languages, in some cases reducing by half.

More interestingly, the relative performance order of the languages is changed. PoS separation for Czech outperforms that of the other languages. Actually, PoS separation for Czech on EuroParl data (Figure 5.2) is scored higher than that of Czech on Universal Dependencies (Figures 5.1c and 5.1d). The source of this improvement requires more study for a proper attribution, but could be due to “beneficial” noise introduced by the automatic tagging, or due to the introduction of more sentence structure by human translators.

The fact that even at low values of $k$, for all languages studied, on both corpora, Exchange Word clusters (and also Brown word clusters, see supplementary material) can successfully separate by Parts of Speech, helps understand why word clusters have had such success at PoS tagging whether coupled with Markov Models [39], Markov Models and morphological features [94], or just by themselves via M-1 [8].

The relation between AMI and PoS

Neither Exchange, nor Brown are guaranteed to converge to a global optimum. Both are greedy algorithms that optimize for high Average Mutual Information (AMI). As we have mentioned earlier, word clusters resulting from Exchange outperform those induced using the Brown clustering algorithm in terms of both AMI (the algorithm’s optimization goal), PoS purity and Adjusted Mutual Information (AdjMI). A natural question to ask is: can one improve the morphosyntactic content of word clusters by obtaining higher AMI, maybe by developing new and better AMI-based clustering algorithms?
5.4. EXPERIMENTS

We answer this question by studying the correlation between Average Mutual Information and the two cluster quality measures used earlier: purity and AdjMI. Brown clustering is a predictable, bottom-up, agglomerative, hard clustering algorithm that for the same hyper-parameter $k$, generates the same clusters and therefore only one data sample. However, the Exchange algorithm is an iterative clustering algorithm that has a complete and valid cluster partitioning at the end of each iteration. Thus, we can also measure morphosyntactic content in each of these clusterings. In our experiments, we only obtain 10 different data samples from each run of the algorithm, not enough for a correlation analysis.

In order to collect more data samples (i.e. more clusterings), we suggest using a stochastic version of Exchange where a percentage of all swaps are performed at random, rather than with the goal of improving AMI. This version of Exchange terminates based on the number of iterations, and generates valid word partitionings of varying quality (from an AMI perspective) at the end of each iteration. In this manner, it provides us with more data points (i.e. more different clusterings) for analysis. Due to the small amount of random swaps, at varying AMI, we obtain a sufficient number of distinct clusterings to perform a correlation study with sufficient data.

With the stochastic implementation of Exchange, we run 50 iterations for all languages and $k$ combinations studied earlier. In Tables 5.2 and 5.3, we show the Pearson and Spearman correlation coefficients between AMI of all clusterings generated by StochasticExchange for a given run, and the two scores used earlier: purity and AdjMI. Due to space considerations, we only show results for $k = 18$ (i.e., same number of clusters as the number of PoS tags). Correlation coefficients for other combinations are included in the supplementary material. $p < 0.01$ for all corre-

---

2 Assuming a stable and repeatable tie-breaking process; this is undefined in the literature.
Table 5.2: Correlation between Average Mutual Information and PoS purity of the clustering resulted from Exchange with $k = 18$. Words with frequency $< 5$ have been filtered. $p < 0.01$ for all coefficients.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Pearson</th>
<th>Spearman</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN UD</td>
<td>0.9776</td>
<td>0.7173</td>
</tr>
<tr>
<td>FR UD</td>
<td>0.9863</td>
<td>0.3976</td>
</tr>
<tr>
<td>CZ UD</td>
<td>0.9883</td>
<td>0.7378</td>
</tr>
</tbody>
</table>

Table 5.3: Correlation between Average Mutual Information and AdjMI of the clustering resulted from Exchange with $k = 18$. Words with frequency $< 5$ have been filtered. $p < 0.01$ for all coefficients.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Pearson</th>
<th>Spearman</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN UD</td>
<td>0.9897</td>
<td>0.7464</td>
</tr>
<tr>
<td>FR UD</td>
<td>0.9845</td>
<td>0.7192</td>
</tr>
<tr>
<td>CZ UD</td>
<td>0.9859</td>
<td>0.8930</td>
</tr>
</tbody>
</table>

For both purity and AdjMI, there is a strong Pearson correlation between higher AMI and better values of the evaluation score. This is independent of the language studied or the number of clusters derived. For Spearman, except for one case, in all combinations studied, there is a high correlation, although to a slightly less extreme degree as with Pearson.

Our experiments show that there is strong correlation between AMI and performance in separation of Parts of Speech as measured by purity and AdjMI. The strong correlation provides grounding for research into new AMI-maximizing word clustering algorithms that can achieve higher AMI than Exchange, or Brown, as such algorithms might be able to separate Parts of Speech even better.

**Effect of polysemy on cluster purity**

In previous sections, we studied the ability of word clusters to encode morphosyntactic information. We clustered word types from unstructured text, where each token had its Part of Speech tag appended. The post-pended PoS tags are not used by either Brown, or Exchange. They are essentially invisible to the algorithms, since the both Brown and Exchange recognize words exclusively by internally assigned integer IDs and do not operate at character level.

However, post-pending PoS tags does introduce some information into the text by providing PoS-role disambiguation for each word occurrence. For example, without post-pended PoS tags, both Exchange, and Brown algorithms, would conflate the two
5.4. EXPERIMENTS

distinct grammatical roles of *show* in the sentence: “Everyone must show their show tickets at the entrance”. In this section, we study PoS separation effects caused by such polysemy on Brown and Exchange word clusters.

Both Exchange and Brown construct hard clusters, i.e. each word can be assigned to exactly one word class. Thus, words with multiple roles, such as denominal verbs or deverbal nouns, cannot be differentiated by the algorithms when operating on corpora from languages where the such morphological derivations are performed without employing suffixes or prefixes. In other words, if the lexical form does not change, neither Brown nor Exchange can identify which tokens represent what grammatical role.

The extent of this effect is dependent on language. In English, for example, nouns are often turned into verbs without changing the lexical form through morphological derivation, e.g. *show* as a verb vs *show* as a noun. On the other hand, Czech is highly inflected accounting for gender, case, number and person. This property of each language was not problematic in the experiments we have performed so far due to the fact that post-pending the PoS tag from ground-truth (or automatic tags) effectively provides disambiguation of grammatical role. Measuring on the Universal Dependencies corpora, we find that the percentage of polyclass words (i.e. word types that are assigned more than one PoS class tag throughout the corpus) varies by language and increases (as percentage of remaining vocabulary) as we raise the minimum frequency threshold, see Table 5.4. For English and French, up to 43% of the vocabulary words have more than one tag, while only 5.5% of the Czech vocabulary shares the same property. Part of the reason why so many words have multiple PoS tags has to do with how the various language families derive new words, and part of the reason stems from errors in PoS tagging of large text corpora [110].

From a practical point of view, polysemous words create an upper bound on the effectiveness of hard clustering for Part of Speech separation (PoS). In Figure 5.3, we show PoS purity for clusters induced over Universal Dependencies (UD) corpora, where we consider all polyclass words as clustered incorrectly. We also show the minimum purity (when $k = 1$) as well as the upper bound given by the polysemy of each language as observed from the manual labels. The evaluation strictly penalizes multiclass polysemy and ignores errors in labeling, such as those identified by Silberztein [110]. For example, in the UD English corpus, even though only 3 occurrences of the word “them” are incorrectly tagged as adverb, while the remaining 750 are correctly labeled as pronoun. We defer to the data and consider the word to be impossible to correctly allocate to a cluster. We use such a strict evaluation as it provides a lower bound on what can be expected from Exchange and Brown clusters given the current data. Correcting PoS tags in the data would probably improve PoS separation, however, such corrections are outside the scope of the work in this paper.

We should point out that this evaluation is not representative of the expected PoS tagging performance of word clusters on any given corpus, as for such taggers one would employ a different strategy, such as, for instance, always outputting the most popular PoS tag for any given word type. On top of that, our evaluation here does not take into account the frequency of tokens, which would be highly relevant for PoS
CHAPTER 5. MORPHOSYNTACTIC CONTENT OF BROWN CLUSTERS

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Min 1</th>
<th>Min 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>EN UD</td>
<td>15.39</td>
<td>43.02</td>
</tr>
<tr>
<td>FR UD</td>
<td>9.09</td>
<td>41.04</td>
</tr>
<tr>
<td>CZ UD</td>
<td>1.81</td>
<td>5.51</td>
</tr>
</tbody>
</table>

Table 5.4: Percentage of vocabulary with multiple PoS tags. Values are calculated relative to the vocabulary remaining after application of a threshold.

![Diagram](image)

Figure 5.3: Cluster purity for manually annotated corpora from UD. Only words with frequency minimum 5. Dotted lines are baselines for $k = 1$ and highest achievable purity given polysemy in corpus.

tagging performance, but not for our evaluation.

As expected, the most affected language is English, due to the high level of polysemy in the data. Here purity drops from 72.4 to 42.32 for $k = 18$, when compared with results in Figure 5.1c. It is followed by a 20 point drop for French, and only a few points for Czech, the most morphologically rich of the three and with the least amount of ambiguity in grammatical role. The results suggest that even in the presence of language ambiguity, and considering the strictest evaluation, Exchange and Brown clusters successfully encode a considerable amount of morphosyntactic information, which varies by language. These, together with results presented earlier in this paper provide empirical evidence for using word clusters as word representations in downstream NLP systems addressing tasks that rely on morphosyntactic knowledge of the language targeted (e.g. dependency parsing), or for use in new paradigms such as data programming [101], where cluster membership can be a strong signal for probabilistic data labeling, even when considering language ambiguity.

5.5 Conclusion

In this paper, we quantified the amount of morphosyntactic information encoded in Brown and Exchange word clusters, in a number of languages, from different language families. Our empirical quantification helps explain the success of word clus-
5.5. CONCLUSION

ters as word representations in NLP tasks that rely on morphosyntactic information, such as PoS tagging and Named Entity Recognition. It further provides empirical evidence for using word clusters as word representations in other NLP tasks that require morphosyntactic knowledge of the language targeted (e.g. dependency parsing), or for use in new paradigms such as data programming [101], where cluster membership can be a strong signal for probabilistic data labeling. We have also shown that there is a strong correlation between AMI (Brown and Exchange’s optimization goal) and performance in PoS separation. The strong correlation demonstrated provides grounding for research into new AMI-maximizing word representation algorithms that can achieve even better AMI optimization than Exchange or Brown.
Chapter 6

Accelerated High-Quality Mutual-Information Based Hierarchical Clustering

Abstract

Word clustering groups words that exhibit similar properties. One popular method for this is Brown clustering, which uses short-range distributional information to construct clusters. Specifically, this is a hard hierarchical clustering with a fixed-width beam that searches greedily for minimized global mutual information loss. The result is word clusters that tend to outperform or complement other word representations, especially when constrained by small datasets. However, Brown clusters tend to be slow to induce, and to find local maxima. This paper shows that replacing the pre-tree stage of Brown with Exchange clustering yields faster results, with better clusterings. Further, one may switch from Exchange to Brown at an arbitrary point, giving control over quality while also affording the benefits of Brown clustering. In addition, a stochastic element can be readily introduced into Exchange. This is demonstrated to enable escaping local maxima, thus finding better global optima through model selection.

6.1 Introduction

Word clusters have been successfully used in NLP tasks over the past decades, contributing to advances especially in machine translation [5, 18], named entity recognition [100, 102], parsing [56, 57], and processing noisy text [95]. They remain a competitive representation useful for many tasks (e.g. [41], [24], [69]), yielding superior extrinsic performance in particular when limited data is available [98] – which is the case for the majority of languages. Brown clustering [17] is a commonly-used word clustering algorithm, performing a hard hierarchical clustering of words based on the global objective of minimized Average Mutual Information (AMI) loss. This tension between local merges and global distance function make the algorithm hard
to parallelize. At the same time, the number of merges considered at any one time directly affects the quality of the final word clustering. While Brown clustering often provides a useful grouping of items into classes based on their distributionality, we find that this process is both slow and also finds poorer optima than using Exchange clustering to do the same. We demonstrate that Brown and Exchange can be combined, to yield better quality clusterings, in less time, while retaining the tree-based features of the generally slower and more involved Brown method. We further retro-fit Exchange with stochastic merging, to allow escape from local maxima.

6.2 Background

We cover Brown and Exchange Clustering from an algorithmic point of view, describing their behavior and how they interact with their objectives, in order to provide an informed evaluation later.

Brown Clustering

During Brown clustering, class pairs are repeatedly merged, with each merge being the one that reduces global average mutual information the least. The initial state is each word type having its own class, and the final state is with all word types in one single class. Each merge alters class-to-class mutual information. The use of a global objective function with no local distance measures makes this slow to compute and difficult to parallelize. To reduce computational times, often only a subset of merges are considered; in the original paper this was merges between the top 1000 classes. This number was later abstracted to a variable $c$. The process is akin to doing beam search for optimal merges, where $c$ is the width of the beam. Later this was formalized as Generalized Brown [38], which also included a method for decoupling cluster generation from merge generation, based on the intuition that each run creates many intermediate clusterings of size 1 to the size of the vocabulary $|V|$, and so to re-create a clustering one can simply re-run the desired number of merges. This then re-purposes beam width as a clustering quality factor, with higher beam widths giving better quality clusterings at the cost of time, running in $O((|V| - c)c^2 + c \log c)$.

The AMI that forms the objective function of both mutual information clustering methods also acts as an intrinsic performance measure. When clustering is performed with a subset of global state being considered at every step, e.g. running wcluster [64] with $c < |V|$, AMI rises monotonically as a greater proportion of the vocabulary is represented in the classes built so far (directly observable in the generalized form’s mergefile output). AMI peaks at the first point where all words are present in the set of classes being considered, and subsequently decreases during tree building phase. We use peak AMI as an intrinsic measure of a clustering run’s performance.
6.3. EXCHANGE CLUSTERING

Algorithm 1 Exchange clustering

1: Initialise \( c \) empty clusters as \( C \)
2: Assign every word \( w \) in vocabulary \( V \) from corpus \( D \) to a cluster
3: Iteration count \( i = 0 \)
4: while Stopping condition not met do
5: \hspace{1em} for word \( w_i \) do
6: \hspace{2em} Calculate best target cluster \( C_t \) for \( w_i \)
7: \hspace{2em} Move \( w_i \) to \( C_t \)
8: \hspace{1em} Increment \( i \)

6.3 Exchange Clustering

This mutual information-based clustering method seeks to generate a hard clustering of words from a corpus while optimizing a maximum likelihood goal [53]. Like Brown and a first-order HMM, it uses a two-sided model \( P(w_i|w_{i-1}) = P(w_i|c_i)P(c_i|c_{i-1}) \), where the class of the predicted word is conditioned on the class of the previous word [37]. The target number of clusters is pre-specified and each word type initialised in a different cluster according to some heuristic. For each iteration, words are examined in order (e.g. descending corpus frequency), and each word moved to the cluster which gives the highest maximum likelihood estimate based on relative frequencies (Algorithm 1). The MLE function for Exchange, from [53], is essentially the following with the addition of Laplace smoothing:

\[
F'_{ML} = \sum_{c_1, c_2} N(c_1, c_2) \ln N(c_1, c_2) - 2 \sum_c N(c) \ln N(c)
\]  

(6.1)

This runs in \( O(c^2|V|n) \), where \( n \) is the number of iterations. Exchange moves words from one cluster to another with the aim of minimizing the perplexity of the underlying class bigram language model on the provided corpus. Martin et al. [72] showed that starting from the log likelihood optimized for by Exchange, through a few simple rewrites, one arrives at Average Mutual Information, the optimization goal of Brown Clustering. A later simplification of Exchange leads to the partially lexicalized predictive Exchange [120].

An early version of Exchange clustering is mentioned in the original Brown clustering paper [17], being performed after the first stage of the method in order to improve AMI, before the tree-building phase; this is not present in any of the Brown implementations that we are aware of. Stopping criteria are not explicitly defined for Exchange; in Section 6.5 we present two intrinsically-based stopping criteria implemented in our code. There are advanced versions of Exchange, such as BIRA [36], which also address some of the issues highlighted here; though we focus on co-
Chapter 6. Accelerated Hierarchical Clustering

<table>
<thead>
<tr>
<th>Peak AMI</th>
<th>Time (s)</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>0.6256</td>
<td>9.998</td>
</tr>
<tr>
<td>Hybrid</td>
<td>0.7087</td>
<td>4.006</td>
</tr>
<tr>
<td>Baseline</td>
<td>1.188</td>
<td>10.93</td>
</tr>
<tr>
<td>Hybrid</td>
<td>1.322</td>
<td>6.447</td>
</tr>
<tr>
<td>Baseline</td>
<td>1.803</td>
<td>20.19</td>
</tr>
<tr>
<td>Hybrid</td>
<td>1.845</td>
<td>7.384</td>
</tr>
<tr>
<td>Baseline</td>
<td>2.108</td>
<td>49.05</td>
</tr>
<tr>
<td>Hybrid</td>
<td>2.137</td>
<td>7.384</td>
</tr>
<tr>
<td>Baseline</td>
<td>2.504</td>
<td>154.7</td>
</tr>
<tr>
<td>Hybrid</td>
<td>2.521</td>
<td>84.71</td>
</tr>
<tr>
<td>Baseline</td>
<td>2.961</td>
<td>678.8</td>
</tr>
<tr>
<td>Hybrid</td>
<td>2.972</td>
<td>302.1</td>
</tr>
</tbody>
</table>

Table 6.1: Peak AMI value with varied beam width / cluster count c. Time is wall-clock seconds.

opting Exchange to enable hierarchical Brown clustering, whereas other variants of Exchange provide only one flat clustering of words as output.

Hybrid Clustering

To generate a hybrid clustering, one first runs Exchange over the data, giving a set number of clusters c. One then runs Brown clustering over this output clustering, taking as input the resulting clusters C and also the source corpus. As Brown treats words and clusters the same, as classes, the initial state can be the output of an Exchange clustering without modifying Brown. While the original Brown clustering paper [17] treats the process as two stages – clustering and tree-building – the generalized form re-casts this as the same process, the difference being that the number of classes being processed has shrunk below the beam search width at the point of transitioning from clustering to tree-building. This is also the point of peak AMI, and the point at which we switch from Exchange to Brown. Thus, for a hybrid clustering, one defines a set number of classes c, runs Exchange over the corpus with these classes, and then runs Brown over the result.

6.4 Experimental Setup

The baseline is a traditional Brown clustering using the implementation of Liang [64], wcluster. This is run with min-occur set to 1, in order to preserve information and provide a fair, higher-AMI clustering that incorporates a maximum amount of corpus knowledge.

For this study we focus on English, using one million words from Reuters Corpus [103]. The experiment machine had dual Intel 8176 and 512GB RAM.

To avoid further AMI loss, instead of “shearing” clusters at fixed bit depths, we use roll-up feature generation [38]. Brown cluster trees are asymmetrical, so when
6.5. RESULTS AND ANALYSIS

Table 6.2: Peak AMI for Exchange and Brown over the complete RCV1 dataset, varying \(c; i = 10\).

<table>
<thead>
<tr>
<th>(c)</th>
<th>Baseline</th>
<th>10</th>
<th>40</th>
<th>160</th>
<th>320</th>
<th>640</th>
<th>2560</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.5656</td>
<td>1.143</td>
<td>1.671</td>
<td>2.154</td>
<td>2.631</td>
<td></td>
</tr>
<tr>
<td>Hybrid</td>
<td>0.6949</td>
<td>1.271</td>
<td>1.747</td>
<td>2.189</td>
<td>2.650</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.3: Extrinsic F1 on CoNLL 2003 NER.

<table>
<thead>
<tr>
<th>(c)</th>
<th>Baseline</th>
<th>10</th>
<th>40</th>
<th>160</th>
<th>320</th>
<th>640</th>
<th>2560</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>72.86</td>
<td>73.41</td>
<td>73.99</td>
<td>74.72</td>
<td>74.60</td>
<td>74.32</td>
</tr>
<tr>
<td>Hybrid</td>
<td>72.44</td>
<td>74.04</td>
<td>74.08</td>
<td>74.98</td>
<td>75.35</td>
<td>74.16</td>
<td></td>
</tr>
</tbody>
</table>

6.5 Results and Analysis

Results are presented in Table 6.1. Here we see that our hybrid method provides a higher peak AMI and lower run-time than the baseline Brown clustering algorithm, in every case. The gap in AMI closes as the number of clusters generated rises. This behavior is preserved in larger corpora. Taking the full RCV corpus of 114M tokens, we observe a similar rise in peak AMI, see Table 6.2.

Extrinsic results are shown in Table 6.3, which indicates improved performance in external tasks, reflecting the higher intrinsic performance.

Stopping criteria

The point at which Exchange is stopped is not well-defined in its original presentation. We propose two simple stopping criteria.

The first is a cut-off for gradient over iterations. If the relative MI increase dips below a given threshold between iterations, the algorithm stops. This bound can be specified regardless of the input data size (which drives the absolute MI values). The box plot in Figure 6.1 shows the percentage of final AMI achieved in each of the first 10 iterations of Exchange, measured over 5 runs with different values for the desired number of clusters (18, 50, 100, 500, 800), using the English Universal Dependencies 2.1 [59] as input data. The vast majority of the final AMI is achieved within three or four iterations. This conclusion is similar to that of Dehdari et al.
[36], who noticed that for small numbers of clusters, the BIRA algorithm (a modified version of Exchange) converges typically within three iterations.

Further, after the first few iterations, only a small proportion of the vocabulary continue to swap clusters (Figure 6.2). Thus, the second stopping condition is the number of words moved in an iteration. Observing the process, many words will find an optimal cluster, regardless of the distribution of most of the other words. We attribute this to the Zipf-Mandelbrot distribution of word frequencies [78] and the effect frequency has on each word type’s contribution to global mutual information. The majority of words share very little MI with most other words. Therefore, the next stopping criterion is based on the number of words that move in an iteration: if this falls below threshold $m$, the clustering stops.

These AMI progression and swapping behaviors can also be observed when clustering the French and Czech corpora from UD. Additional data is in the supplementary material.

### Stochastic Merging & Model Selection

Exchange applies a greedy method, the output of which depends on both the prior state and the order in which words are examined, leaving little opportunity for escaping local maxima. Stochastic swapping with chance $r$, with cluster assignment

<table>
<thead>
<tr>
<th>$c$</th>
<th>English UD</th>
<th>French UD</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>0.9950</td>
<td>1.3079</td>
</tr>
<tr>
<td>50</td>
<td>1.3183</td>
<td>1.8162</td>
</tr>
<tr>
<td>100</td>
<td>1.5960</td>
<td>2.1472</td>
</tr>
<tr>
<td>500</td>
<td>2.3706</td>
<td>2.8240</td>
</tr>
<tr>
<td>800</td>
<td>2.6897</td>
<td>3.0635</td>
</tr>
</tbody>
</table>

Table 6.4: Peak AMI with stochastic merging. $i = 50$

Figure 6.1: Box plot of per-iteration Exchange AMI as percentage of final AMI. The whiskers are the maximum and minimum values.
6.6. Conclusion

This paper shows distributionally-derived hierarchical word clusterings can be induced with improved speed and at high quality by combining Exchange and Brown clustering, compared to using Brown alone. Further, it introduces a naïve but effec-
tive method for avoiding local maxima during Exchange clustering, leading to other performance boosts. C++ code for the hybrid tool is made available with this paper.
Abstract

Abbreviations often have several distinct meanings, often making their use in text ambiguous. Expanding them to their intended meaning in context is important for Machine Reading tasks such as document search, recommendation and question answering. Existing approaches mostly rely on manually labeled examples of abbreviations and their correct long-forms. Such data sets are costly to create and result in trained models with limited applicability and flexibility. Importantly, most current methods must be subjected to a full empirical evaluation in order to understand their limitations, which is cumbersome in practice.

In this paper, we present an entirely unsupervised abbreviation disambiguation method (called UAD) that picks up abbreviation definitions from unstructured text. Creating distinct tokens per meaning, we learn context representations as word vectors. We demonstrate how to further boost abbreviation disambiguation performance by obtaining better context representations using additional unstructured text. Our method is the first abbreviation disambiguation approach with a transparent model that allows performance analysis without requiring full-scale evaluation, making it highly relevant for real-world deployments.

In our thorough empirical evaluation, UAD achieves high performance on large real-world data sets from different domains and outperforms both baseline and state-of-the-art methods. UAD scales well and supports thousands of abbreviations with multiple different meanings within a single model.

In order to spur more research into abbreviation disambiguation, we publish a new data set, that we also use in our experiments.
7.1 Introduction

Abbreviations are shortened forms of phrases or single words, employed most often in written language. When the text denoting a concept is long and an author must refer to it multiple times, it is easier for the author to only use the abbreviation. Thus, abbreviations are many-to-one mappings from long-forms to short-forms, used when space or human limitations make the short-form more convenient. Due to the many-to-one mapping, abbreviation usage can cause problems for automated, computer-based, readers and represents a challenge for various natural language understanding tasks. For example, PCB is an abbreviation for Polychlorinated biphenyl, Printed Circuit Board, Pakistan Cricket Board and a few other concepts. Without domain knowledge, the meaning of PCB in the sentence: The environmental measures taken include limiting the of substances such as cadmium, lead, PCB, and Azo pigments, cannot be correctly inferred to be Polychlorinated biphenyl. Abbreviations are particularly prevalent in biomedicine literature, often confusing readers [45]. In MEDLINE abstracts, a high percentage of abbreviations can have multiple meanings (64.6%) [67]. Structured knowledge bases such as the Unified Medical Language System (UMLS) also contain a considerable amount of ambiguous abbreviations (33.1%) [68].

Resolving the intended meaning of an abbreviation in a given context is crucial for search engines, question answering systems, document recommendation systems, or text analytics. All systems addressing Natural Language Understanding tasks are expected to automatically know which concept a sentence refers to, even if only an abbreviation (i.e., short-form) is used to denote said concept. If a search engine is given a query like effects of Polychlorinated biphenyl on the environment, one would expect the search engine to return documents referring to Polychlorinated biphenyl either with its long-form or with its short-form. Similarly, recommender systems suggesting further literature on the topic of Printed Circuit Boards should not suggest only documents where the authors employ exclusively the long-form, but also those where the short-form PCB is used to denote the same meaning. Indeed, search engines can return different results if queried with abbreviations or long-forms [43], and this problem can be compounded by ambiguity.

Previous abbreviation disambiguation methods have relied on corpora of manually annotated meanings and used hand-designed features [79, 80, 123, 124]. Such approaches have two major weaknesses. First, the cost of annotating abbreviation meanings is high as it involves human labor, often from domain experts. Secondly, models trained on such corpora can only disambiguate abbreviations that have been manually annotated. In order for such systems to support new abbreviations, more human work from domain experts is necessary to create more annotations, leading to higher costs. Similarly, if disambiguation is needed for a new domain, different domain experts are required to first manually annotate a corpus. Human annotation costs have prevented the scaling up of abbreviation disambiguation to large collections of abbreviations spanning multiple domains. Indeed, most data sets for abbreviation disambiguation are quite small [15, 63, 83, 124].
Our method is completely unsupervised. It automatically extracts short-forms and their possible long-forms from large corpora of unstructured text. Thus it can learn what abbreviations exist and their possible meanings based on the provided unstructured corpus. We learn a different word vector for each long-form, and for all context words, in order to create semantic representations of the possible long-forms and the context in which they are used. Using this representation space, our method can easily identify the most likely intended long-form for an ambiguous abbreviation based on the context in which each abbreviation meaning is used. As the first method in this field, we provide an analysis of the expected disambiguation performance that does not require costly and cumbersome empirical evaluation studies on manually annotated test data. Additionally, we introduce a technique that can boost the quality of the context representations using external text corpora for learning. In summary, our method is completely unsupervised, generally applicable, flexible in terms of application domains and text content, as well as easy to review and update.

In this paper, we present the following contributions: (1) the Unsupervised Abbreviation Disambiguation (UAD), a disambiguation method that is unsupervised and does not employ hand-designed features, (2) a data set for evaluating performance on the abbreviation disambiguation task. For paper review purposes, the data set and evaluation scripts are available at http://bit.ly/2DTkMh7. They will be moved to the Harvard Open DataVerse before camera ready.

### 7.2 Related Work

Supervised approaches to abbreviation disambiguation require sense-annotated data sets, usually annotated manually, by domain experts. These methods often rely on hand-designed features such as co-occurrence (e.g., with other words in the corpus), number of uppercase letters etc [79, 80, 123]. These features are usually designed using domain knowledge or are language-specific. Our method does not use hand-designed features.

One approach to avoiding hand-designed features is to use word vectors to represent words and long-forms. A short paper by Li et al. [63] briefly sketches this idea and presents two models built using word2vec vectors [75]. Both models represent long-forms using a single vector calculated as a weighted average of vectors of all words observed in the context of each long-form, in all training examples. The two models differ in the underlying word2vec model and the weighing strategy. The first model, TBE, averages CBOW-derived word vectors of surrounding words using TF-IDF scores. It also limits the terms used in the average computation to the top $n$ in the window surrounding a long-form usage. The other, SBE, represents context as the sum of vectors of contextual words observed in all occurrences in the training data, but uses Skip-gram-derived word vectors. Thus, for both models proposed by Li et al., vectors for long-forms are calculated as an average of surrounding context words, given all observations of each long-form. The central idea is to create a fingerprint of the context that matches each long-form by averaging vectors of all contexts.
in which the long-form was observed. Once vectors have been computed for each long-form, disambiguating an abbreviation use reduces to picking that long-form candidate whose vector has the smallest cosine distance to the average of vectors of words surrounding an abbreviation use. Li et al. show experimentally that the SBE model outperforms TBE. More recently, Charbonnier and Wartena [22] present Distr. Sim., a variation of TBE that replaces weighing by TF-IDF with weighing by IDF only. Distr. Sim. is based on the same idea of creating context fingerprints as TBE. In fact the only difference is the weighing strategy. Unlike these two methods, in our model, vectors for long-forms are not fingerprints of observed contexts, but trained in relation to their context, by a language model, at the same time as the rest of vector space. Thus, their value is established by the language model trained by word2vec, and their vectors’ values influence those of contextual words around each long-form usage.

One problem affecting unsupervised abbreviation disambiguation is that of identifying valid long-forms. Many unsupervised methods identify valid long-forms by first identifying definitions of abbreviations. Such methods must account for inconsistencies in writing long-form definitions. In the biomedical domain, Okazaki et al. [93] found that many long-forms represent the same sense even when lexically different, e.g., pathologic complete response and pathologically complete responses. They propose a supervised clustering method, which normalizes the long-forms into clusters of senses using a similarity measure trained on term variations observed by a domain expert. Disambiguation is based on hand-designed features built from unigrams and bigrams. Experiments show the value of clustering long-forms and limiting their variations. In contrast, the normalization step used in our method is simpler and does not require any labeled data.

There have been some attempts to formulate abbreviation disambiguation as a classification problem. Since the set of labels (i.e., long-forms) is specific to each short-form, Wu et al. [124] express abbreviation disambiguation as a series of classification problems, one for each abbreviation (i.e., short-form). Their system combines word vectors with hand-designed features and uses a Support Vector Machine to train a classifier for each supported short-form. However, only distinct 75 abbreviations are included in the system. Such a method is impractical for industrial settings that require support for thousands of distinct abbreviations with, potentially, tens of thousands of long-forms. By contrast, our system builds a single model that allows disambiguation of all abbreviations and does not use hand-designed features, making it generally applicable, including for multiple languages.

Entity Linking (EL), often referred to as Record Linking, Entity Resolution or Entity Disambiguation, is a Natural Language Processing task that addresses ambiguities in text and bears some similarity to Abbreviation Disambiguation. The EL task consists of identifying entity mentions in text and providing links from the mentions to knowledge base entries. Entity Linking can target named entities (i.e., names that uniquely refer to entities of predefined classes such as person, location or organization), or common entities (also referred to as Wikification) where all noun phrases determining entities must be linked to concepts in a given knowledge base.
The choice of entities to disambiguate depends on downstream tasks which can, for example, be to track named entities for document indexing, relation extraction, or question answering [66, 108].

Both Abbreviation Disambiguation and Entity Linking, establish connections between a lexical form and one of potentially many meanings. Despite this similarity, the two tasks differ in fundamental aspects. In Entity Linking ambiguity, at least in part, is due to partial forms which exist due to pre-established context (e.g., referring to George Washington as George, or Washington), metonymy (e.g., using the name of a country or its capital city as a substitute for its government), or entity overlap (e.g., mentioning a city together with its geographic location, such as Madison, Wisconsin) [66, 82, 99]. Often, in Entity Linking, the disambiguation task is simplified by the possibility of finding the correct linking via co-reference resolution.

Another core difference lies in the fact that Entity Linking is based on few distinct types (traditionally place, organization, person) that are insufficient to differentiate between different meanings of abbreviations [99]. More importantly, Entity Linking is a supervised task and requires ground truth, entity types for proper linking, as well as access to a knowledge base. In Abbreviation Disambiguation such types are not available and are not practical due to the large number of types and their granularity. And, as mentioned before, our method is entirely unsupervised, and does not require a knowledge base.

### 7.3 Method

In this section, we present Unsupervised Abbreviation Disambiguation (UAD), a method that requires only a corpus of unstructured text from which it extracts examples of abbreviation uses together with the intended meanings. Because of its simple input requirements, UAD can scale to large corpora and can easily be adapted to new domains by providing it with a corpus of unstructured text from the new domain. UAD exploits properties of *word vectors* in order to represent the dependency between long-forms and the contexts in which they are used. UAD does not rely on hand-designed features.

Input to *Abbreviation Disambiguation* can be formalized as a triple \((s_x, L_{s_x}, u)\), where \(s_x\) represents a short-form, \(L_{s_x}\) is the set of all known long-forms associated with short-form \(s_x\), and \(u\) is a sentence where the short-form \(s_x\) is used without a definition. An abbreviation disambiguation system should output the long-form \(l_y \in L_{s_f_x}\) that represents the meaning of \(s_x\) intended by the author in the sentence \(u\). In our experiments, we only focus on *ambiguous abbreviations* (i.e., short-forms \(s_x\) for which \(|L_{s_x}| \geq 2\)). We do not consider *unambiguous abbreviations* (i.e., those \(s_x\) for which \(|L_{s_x}| = 1\). Unambiguous abbreviations can be expanded using a dictionary lookup as their expansion does not involve any form of disambiguation [28]. In the Unified Medical Language System, a popular ontology in the field of medicine, 76.9% of abbreviations are unambiguous, and thus, require no disambiguation [68].
CHAPTER 7. UNSUPERVISED ABBREVIATION DISAMBIGUATION

Unsupervised Disambiguation

UAD is based on the idea of learning to represent words such that context can be used to predict the presence of often co-occurring words. We use this idea to express the disambiguation task as a prediction problem. An overview of the UAD pipeline is presented in Figure 7.1. The following sections describe each stage in detail.

Example extraction

We start by using a large corpus of unstructured text to extract sentences containing short-forms whose intended meaning is known (stage 1 in Figure 7.1). We call these sentences unambiguous uses of abbreviations since the intended expansion is defined locally in the text. Unlike previous work, we do not rely on human-annotated data sets to learn mappings. Instead, we extract them from text using an extension of the patterns defined by Schwartz and Hearst [106]. The core idea expressed by the patterns is that abbreviations are often defined using the pattern long-form (short-form) or short-form (long-form). For example, a printed circuit board (PCB) is a support for electronic components that also provides physical conductive connections is an unambiguous use of the abbreviation PCB meaning Printed Circuit Board. From this sentence, we extract the tuple (sentence, PCB, Printed Circuit Board). The patterns also define character matching rules between the short-form and long-form that filter out most occurrences of the pattern that are not definitions. For example from the sentence “Memory cards (SD) are used for data storage”, is ignored as it contains no abbreviation definition. Our implementation of the long-form mapping implementation has an accuracy of 86.13% on abbreviations in the Medstract data set [118] and 62.48% on those in Yeast [106]. For details on the mapping algorithm and pseudocode, we refer the reader to the original paper [106]. The performance difference between our implementation and the original is due to our stricter definition of abbreviations.

We identify abbreviations using regular expressions, and support single token abbreviations starting with a capital case letter and consisting of upper and lower case letters, digits, periods and dashes, but consisting of at least 60% capital letters. Our implementation of the patterns proposed by Schwartz and Hearst follows the description closely and thus, does not allow extra tokens inside parenthesis. As such, it cannot identify some abbreviation definitions such as “Printed Circuit Boards (abbreviated PCB)” or “Printed Circuit Boards (PCBs for short)”. Missing such definitions
is generally not a problem as abbreviations tend to be defined multiple times in large corpora. During this stage of our pipeline the priority is high extraction precision rather than recall. We do not focus on exhaustive extraction of all definitions since the meaning of abbreviations can be learned even if only a subset of each meaning’s occurrences is extracted. Once an abbreviation definition is identified in a document, we assume that its meaning stays fixed in the document, and we also extract the subsequent uses in sentences as examples of usage for our model to learn from.

Thus, we apply a simple method to extract abbreviation usage examples from a large corpus. We do not aim for high extraction recall for definition occurrences, but rather on precision and try to eliminate wrong mapping extractions. Using a large corpus of text makes this strategy viable as abbreviations tend to be defined several times in a corpus, often with different styles. This stage of the UAD pipeline is designed as a module, so that it can be swapped with a better performing module when that is found.

**Long-form normalization**

Long-form normalization (stage 2 in Figure 7.1) follows example extraction and aims to filter noise and establish ambiguous abbreviations (i.e., short-forms that map into more than one long-form). We achieve both of these goals by normalizing long-forms that appear to denote the same meaning.

Long-forms that denote the same meaning can sometimes appear distinct due to inconsistent spelling (British vs. American English), different prepositions in long-forms, inconsistent use of hyphens, spaces, or plurals. For example, in the Wikipedia data, we found every single possible hyphenation of the long-form *White Anglo-Saxon Protestants (WASP)*. To such inconsistencies, we normalize long-forms based on lexical similarity. We first strip long-forms of characters that cause lexical diversity without adding information such as spaces, hyphens, and ending characters that denote pluralism. This removes superficial variation in long-forms (e.g., *Amino Acid* and *amino-acids* are normalized to *aminoacid*).

For each short-form, all stripped long-forms are compared with each other using the Levenshtein string edit distance [60]. We normalize the less frequent long-form to the more popular one, if the ratio between the string edit distance and the length of the longer long-form is low. This is based on the assumption that the correct spellings of long-forms are more popular than incorrect variations. Practically, we express each normalization decision as a rewrite rule so that for each tuple \((l_i, l_j)\), the normalization function \(n\) returns that parameter to which both long-forms should be normalized, or *null* to denote that no rewrite is necessary (i.e., the two long-forms are too distinct). The rewrite rules allow us to chain long-form rewrites so that, for example, we could normalize a long-form \(l_i\) to \(l_j\), and then \(l_j\) to some other \(l_h\).
belonging to the same short-form (i.e., \( l_i, l_j, l_h \in L_{s_x} \) for some \( s_x \)). Formally:

\[
\forall s_x \text{ and } \forall (l_i, l_j) \in \{L_{s_x} \times L_{s_x}\}:
\]

\[
n(l_i, l_j) = \begin{cases} 
\arg\max_{l_i, l_j} freq(l), & \text{if } \frac{\text{lev}(l_i, l_j)}{\max(|l_i|, |l_j|)} \leq t \\
\text{null}, & \text{otherwise}
\end{cases}
\] (7.1)

where \(|l_x|\) is the length of \( l_x \). Threshold \( t \) was set to 0.2 in order to keep rewrites conservative. Results \( \text{null} \) are ignored.

After all rewrites are generated, we follow rewrite chains until we establish canonical long-forms. Normalization removes pseudo ambiguity in the data which allows us to (1) identify and ignore unambiguous abbreviations (i.e., those \( s_x \) for which \(|L_{s_x}| = 1\), (2) identify long-forms that denote distinct meanings, and (3) remove noise from the extraction stage as that can be easily identified as low-frequency mappings from short-forms to long-forms. UAD’s normalization stage is simpler than the logistic classifier clustering used by Okazaki et al. [93] which also requires manually labeled data from which to learn a similarity measure.

**Vector space construction**

To disambiguate abbreviations, we formulate the task as a word prediction task where a model must predict the missing word in a sentence. Thus, in the vector space construction step (stage 3 in Figure 7.1), we take the output from the previous step and reformulate each sentence. We replace all tokens constituting short-forms and their definitions by a single placeholder token representing the short-form and its intended, normalized, long-form. For example, the sentence mentioned above is rewritten to

\[A \text{ } _{-\text{ABB}}/\text{PCB}/\text{printed}_c\text{ircuit}_b\text{oard} \text{ is a support for electronic components that also provides physical conductive connections.}\]

The special \( _{-\text{ABB}}/\text{PCB}/\text{printed}_c\text{ircuit}_b\text{oard} \) token allows us to identify the short-form that was used in the original sentence and its intended meaning. The \( _{-\text{ABB}} \) prefix ensures that the placeholder token does not match any natural language token, and allows us to identify all short-forms and long-forms used in the data. In this example, the word prediction task becomes one of predicting that the special token \( _{-\text{ABB}}/\text{PCB}/\text{printed}_c\text{ircuit}_b\text{oard} \), is the missing token in the sentence \( A \text{ } ____ \text{ is a support for electronic components that also provides physical conductive connections.} \)

To train a model that can solve the prediction task, we turn to word vectors derived using language models, more specifically the method word2vec [75] which derives word vectors that encode information from a language model using a shallow neural network. Word2vec can use two training strategies: Continuous Bag-Of-Words (CBOW) and Skip-grams. When using CBOW, the probability of a word \( w_i \) is estimated using information from a linear context constructed using a window of length \( n \) on both sides of \( w_i \) as:

\[P(w_i|s) = p(w_i|w_{i-1}, \ldots, w_{i-n+1}, w_{i+1}, \ldots, w_{i+n})\]
When training using Skip-grams, a word $w_i$ is used as input, and the model attempts to predict the surrounding words within a window of size $n$. In order to encode the language model, word2vec learns a representation of words in a multi-dimensional continuous space. Word representations can be used as inputs to the neural network in order to predict words according to the language model. Thus, most of the information required for language model prediction is encoded in the word vectors. Since the vectors are computed using a language model on a corpus of unlabeled text, word2vec is essentially unsupervised. However, word2vec requires many word repetitions in order to appropriately estimate a vector for each word [75].

We provide the sentences containing placeholder tokens to word2vec in order to derive word vectors that represent each token in the corpus (including the placeholders). We represent each long-form with a unique placeholder token so that, at training time, each long-form is allocated its own unique vector. The placeholder tokens are represented in relation to their contexts following word2vec’s learning approach. Both placeholder tokens and words that often appear surrounding them, are represented in the vector space in such a way that their relationship is automatically encoded in the vectors using word2vec’s underlying language model. In contrast, both SBE and Dist. Sim. rely on vector compositionality combined with weighing methods to construct representations for long-forms by creating fingerprints of the contexts observed in the data. They effectively ignore word2vec’s underlying language model.

To disambiguate the use of a short-form $s_x$ in a sentence $u$, we first create a vector $c$ to represent the context around $s_x$. The vector $c$ is the average of the vectors corresponding to words in the short-form’s context. The disambiguated long-form is that candidate long-form that minimizes the cosine distance between the context vector $u$ and the vectors representing placeholder tokens for each long-form $l_i \in L_{s_x}$:

$$d(s_x, c) = \arg \min_{l_i \in L_{s_x}} \cosine(c, l_i) \quad (7.2)$$

Using this method, we can disambiguate any number of abbreviations without requiring data annotated by domain experts or hand-designed features. The only requirement is that each long-form is observed a sufficient number of times in the corpus before it can be reliably represented in the vector space. This requirement is not a limitation in practice as a larger unstructured corpus can always be found. The frequency requirement is also characteristic of all methods that rely on word vector spaces trained based on their usage. In our experiments we consider 50 observations to be sufficient for word2vec to construct stable and reliable vector representations. Since new short-forms and long-forms are introduced all the time, approaches relying on traditional, human-annotated data, require new manual annotations in order to support disambiguation of new abbreviations. UAD requires only a large corpus of unstructured text that contains uses with definitions of new abbreviations. Similarly, adaptation to a new domain only requires a large unstructured corpus of domain-specific text.
Global Vectors (GloVe) [96] is a method for learning vector representations for words and is often considered an alternative to word2vec. It aims to combine the benefits of matrix factorization methods (i.e., the use of global corpus co-occurrence statistics, as opposed to only information that is available in the local window) with word2vec’s skip-gram model in order to improve performance of resulting word vectors when used for word similarity, word analogy, or Named Entity Recognition. Both similarity and analogy tasks try to mimic the overall human perception of similarity between words or relations in pairs of words. In principle, GloVe vectors can be used as a drop-in replacement to word2vec in our disambiguation method. However, we expect that GloVe’s prioritization of global co-occurrence to be detrimental to representations of long-forms as a function of context. The core idea in UAD is to allow a predictive language model to learn vector representations of both long-form meanings and words that appear in the context of abbreviations. For abbreviation disambiguation, we expect the words surrounding a short-form or long-form to be of much greater importance for correctly identifying the semantics than overall frequencies or co-occurrence as is the case for similarity and analogy tasks. In Section 7.4, we study experimentally GloVe as a drop-in replacement for word2vec as the word vector derivation method.

Pre-evaluation analysis

In the previous section we presented UAD, an unsupervised method that can learn what abbreviations are present in a corpus, what meanings they have, and how to disambiguate them based on context. In this section, we present a property of UAD that addresses practical aspects of deploying abbreviation disambiguation in industrial settings.

Tailoring abbreviation disambiguation to a specific domain can improve performance by allowing the model to learn abbreviation meanings (i.e., long-forms) characteristic to the domain. Alternatively, one can avoid learning long-forms that are not used in the target domain, thus making disambiguation easier by reducing the number of long-form candidates for each short-form. For example, the Pakistan Cricket Board long-form of the abbreviation PCB used in examples earlier, is improbable to appear in chemical scientific literature. Automatically removing this long-form from the disambiguation model would eliminate the possibility of wrongly disambiguating to that long-form and, in some cases, it will reduce the number of possible long-forms for an abbreviation to 1, thus making the disambiguation process straightforward (i.e., dictionary lookup). UAD is well suited for domain adaptation due to its unsupervised nature.

Usually, after training a disambiguation model on a new corpus (or for a new domain), expensive, extensive evaluation is required to properly understand overall performance, and identify individual problematic cases, such as abbreviations whose long-forms are easily confused. Typically, this is done by repeated cycles of training, evaluation on test cases (for which costly manual labels are needed), followed by gathering, cleaning, and potentially labeling of new data. This iterative process
METHOD

7.3. METHOD

can take considerable effort and time. Unfortunately, such tedious cycles are the norm for adapting existing methods to new domains [79, 123, 124]. In industrial environments, where deployment speed of disambiguation models is crucial, costly, full-scale evaluations on domain data may prove impractical. A method that does not require such expensive evaluation cycles can allow for faster and more flexible deployment in new domains, and for efficient model updates.

For UAD, we propose a novel, straightforward, rapid evaluation of the expected disambiguation quality, e.g., when updating to newly discovered abbreviations, or when tailoring the model to a new domain. Specifically, UAD can provide fully unsupervised insights into performance on target data, without requiring labeled data, or manual creation of test cases. Our pre-evaluation method is based on the observation that issues with abbreviation disambiguation performance translate directly into selecting incorrect long-forms for a given short-form and context. Consequently, we study which pairs of long-forms are difficult to disambiguate based on their context models. We show how this can be done using only the vector representations. If two long-forms belonging to the same short-form have similar representations in the vector space, they are clearly difficult to disambiguate. Let $l_i$ and $l_j$ be two long-forms belonging to the same short-form $s_x$, i.e., $l_i, l_j \in L_{s_x}$. If the vectors of $l_i$ and $l_j$ are similar under cosine distance, then any context vector $c$ that is close under the cosine distance to one of them, will also be close to the other. We only need to consider $l_i$ and $l_j$ which belong to the same short-form $s_x$ as UAD emits long-forms only from the set known to belong to the short-form that requires disambiguation. Thus, pairs of long-forms belonging to different short-forms do not cause problems for disambiguation. Formally, for the set $R_{s_x}$ containing all unordered pairs of long-forms that share the same short-form $s_x$:

$$R_{s_x} = \{\{l_i, l_j\} \mid \exists s_x : l_i, l_j \in L_{s_x}\}$$ (7.3)

we construct a ranking $P_{R_{s_x}}$ of the unordered pairs based on the cosine distance between the members of each pair. In other words, the unordered pair $\{l_i, l_j\}$ is more problematic than the unordered pair $\{l_k, l_l\}$ if and only if:

$$\text{cosine\_dist}(l_i, l_j) < \text{cosine\_dist}(l_k, l_l)$$ (7.4)

In Equation (7.4) we do not require that the two pairs of long-forms $(l_i, l_j)$ and $(l_k, l_l)$ belong to the same short-form $s_x$. The ranking $P_{R_{s_x}}$ can be easily reviewed for problematic long-form pairs without requiring costly evaluation on labeled test cases. The two unordered pairs of long-forms $\{l_i, l_j\}$ and $\{l_k, l_l\}$ need not belong to the same short-form as we are ordering pairs based on how difficult it is for UAD to distinguish between the elements of each pair, as opposed to between the pairs themselves. Our pre-evaluation based on the ranking $P_{R_{s_x}}$ only uses the trained disambiguation model and no labeled data. In section 7.4 Experiments, we measure the correlation between cosine distances of long-forms and disambiguation performance. Existing abbreviation disambiguation methods whose models are opaque to explanations [124], cannot provide such insights into their models, and thus must be subjected to a full-scale evaluation on ground truth, manually labeled data.
Context Representations

Proper context representation is critical to the performance of UAD. In the following subsections, we discuss how to create word vector spaces that effectively capture context.

Augmentation with background text

As described previously, for derivation of vector spaces, UAD is provided with input consisting of only examples of unambiguous usage of ambiguous abbreviations that are automatically extracted from a large, unstructured, corpus. Small and targeted corpora may exhibit relatively high ratios of vocabulary to corpus length which means that words are observed relatively few times in the corpus. This can be problematic for unsupervised word representation methods such as word2vec. The small number of word uses may not provide enough information about the relation of words to their contexts, leading to non-representative vectors. In order to give word2vec a better chance to reach a stable vector space configuration, and to improve disambiguation performance, we augment the corpus used for vector space training. We supplement the set of abbreviation usage examples with an extra corpus of unstructured text. We aim for as high a vocabulary overlap with the abbreviation examples as possible in order to provide maximum word usage information. The addition of background text provides usage information for words that appear in both corpora. Since the location of long-form placeholders is influenced by that of all other words in the vocabulary through context-sharing, the locations of long-form placeholders will also be influenced, leading to overall improvements in abbreviation disambiguation. For maximal benefit, the augmentation text should share the same domain and writing style as the abbreviation examples. This can be achieved by extracting the background text from the same large, unstructured, corpus as the abbreviation examples. In experiments where we use background text, we denote such augmentations with TXT.

Stop-word removal

Tokens that often occur in language use, such as prepositions, articles, punctuation etc. (often called stop-words) can reduce vector space quality. At training time, word2vec adjusts vectors of words surrounding stop-words in order to increase its performance in predicting stop-words. This behavior is undesired since it leads to lower quality vectors for tokens other than stop words. Moreover, our method (UAD) compounds this issue because at disambiguation time, it computes an average vector of all contextual tokens where content-bearing words have the same weight as stop-words.

In order to eliminate such undesired effects, we remove stop-words from the learning corpora. This also has the effect of effectively increasing the window size for word2vec as every stop-word removed allows another token to enter the window, if any is left in the sentence.
7.4 Experiments

Data Sets

We used two data sets for all experiments: one based on the English Wikipedia, and one based on PubMed. All of English Wikipedia as of 1st of August 2017 was downloaded, and all text content was extracted using WikiExtractor [4]. For the PubMed data set, we downloaded the Commercial Use Collection of the Open Access Subset of PubMed [88] and extracted only the text content from each article.

From each corpus, we selected only sentences with unambiguous abbreviation usage, and normalized long-forms using the methods described in Section 7.3. We kept only those short-forms that had at least two long-forms (i.e., ambiguous abbreviations), each of which occurred in at least 50 examples. This minimum frequency threshold was chosen for two reasons. First, all three competitors considered in our experiments rely on vector space representations and are thus susceptible to unstable vectors for low-frequency terms. Second, the frequency threshold acts as a filter that reduces noise due to incorrect mappings during the first stage of our pipeline. All sentences containing uses of ambiguous abbreviations were distributed into 10 bins for the purpose of 10-fold cross-validation. Since we know the intended long-form for each of our extracted and normalized examples, we can perform 10-fold cross-validation without requiring manual labels. In order to provide more in-depth evaluation, we manually annotated a subset of the 10th fold of our Wikipedia data set. This evaluation is discussed separately. The manually labeled data set is published together with the full Wikipedia data set and evaluation scripts.

For every data set, we also extracted a collection of sentences that is used as background corpus to derive higher quality word vectors (the background corpus is denoted TXT in all experiments).

A summary of the data sets is provided in Table 7.1. We created the two data sets used in our experiments to address shortcomings in existing evaluations. Many data sets used in previous studies are too small to provide an accurate image of real-life disambiguation performance and some have artificial biases. For example, the MSH-WSD subset used by Li et al. [63] has a low degree of ambiguity (2.11) and is heavily balanced, i.e., an almost equal number of usage examples are provided for each long-form. We do not consider such data sets sufficient for accurately assessing disambiguation performance. None the less, we have run experiments with the MSH-WSD subset from Li et al. [63] for repeatability, which we include in Section 7.4 below. The data of Charbonnier and Wartena [22] is not publicly available, but we have, of course, also evaluated their method, Distr. Sim. in our experimental study.

Even the smaller of our data sets, Wikipedia, is up to one order of magnitude larger than the ones used in previous literature [15, 63, 83, 124]. Our data sets help provide a better view of disambiguation performance as the data sets contain more varied test and learning examples. The ratio between the number of long-form examples in either of our data sets is not artificially balanced, i.e., they represent the relative popularity of the long-forms as observed in the corpora. The lack of artifi-
special balancing allows for establishing a clear picture of the effectiveness of simple
baselines, such as frequency-based disambiguation. We made the Wikipedia data set
available online together with the evaluation script used in our experiments.

Performance measures

We measured performance as follows. For every long-form we calculated precision
and recall as:

\[
\text{precision} = \frac{TP}{TP + FP} \quad \text{recall} = \frac{TP}{TP + FN}
\]

Where \( TP \) stands for true positives, \( FP \) for false positives and \( FN \) stands for false negatives.

Performance measures are computed using micro-, macro-, and weighted versions of precision, recall and F1. For the purpose of evaluation, we treat the problem as multi-label classification. Thus, micro- versions of precision, recall, and F1 are equal to each other and take the same value as accuracy. Thus, we only report accuracy. Weights for the weighted measures are provided by the observed frequency of each long-form in the learning set. Overall scores are averages of each fold. We calculated both weighted and macro metrics in order to better understand the observed performance. Weighted precision, recall and F1 provide an estimation of real-life performance since abbreviations that occur more often have a higher weight. On the other hand, macro- measures describe how many abbreviations are properly disambiguated as there is no weighing between the different labels. For example, if an experiment leads to an increase in unweighted precision, but a drop in the weighted one, we can conclude that the new version handles less frequent abbreviations better, to the detriment of some frequent ones. The evaluation script that calculates all performance metrics is made available together with the Wikipedia data set.
7.4. EXPERIMENTS

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Acc.</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 FREQUENCY</td>
<td>54.14</td>
<td>30.04</td>
<td>54.14</td>
<td>38.46</td>
<td>25.55</td>
<td>46.34</td>
<td>32.79</td>
</tr>
<tr>
<td>2 SBE [63]</td>
<td>82.48</td>
<td>83.07</td>
<td>82.48</td>
<td>82.53</td>
<td>82.18</td>
<td>82.16</td>
<td>81.87</td>
</tr>
<tr>
<td>3 Distr. Sim. [22]</td>
<td>80.19</td>
<td>80.87</td>
<td>80.19</td>
<td>80.25</td>
<td>79.90</td>
<td>80.12</td>
<td>79.71</td>
</tr>
<tr>
<td>4 UAD with TXT</td>
<td><strong>90.62</strong></td>
<td><strong>92.28</strong></td>
<td><strong>90.62</strong></td>
<td><strong>90.66</strong></td>
<td><strong>91.35</strong></td>
<td><strong>91.36</strong></td>
<td><strong>90.59</strong></td>
</tr>
</tbody>
</table>

Table 7.2: Competitor comparisons on the subset of MSH WSD used by Li et al. [63]

Results

Setup

UAD only requires those hyper-parameters specific to word2vec. More specifically, in all experiments we set \( w=10, i=10, size=300, neg=5 \) and the model to Skip-gram, unless stated differently.

Comparisons with baseline and existing methods

In the following experiments, we compare our method, UAD, with a simple, but efficient baseline, and two state-of-the-art methods: SBE [63] and Distr. Sim. [22]. The baseline (FREQUENCY) outputs the most frequent long-form for each short-form. Its statistics are calculated based on the frequencies observed in the 9 folds available for learning. Thus, the simple baseline FREQUENCY performs no context-based disambiguation and relies exclusively on corpus statistics. We implement SBE [63], and its variation, Distr. Sim. [22], and train both methods using the parameters given in the original papers. We adapt our pre-processing pipeline discussed Section 7.3 together with details in each of the competitors’ papers in order to provide them with the same set of manually extracted examples as we use for our experiments. This evaluation is slightly biased against UAD as some competitors (SBE) are supervised methods requiring manually labeled data.

To evaluate SBE, Li et al. [63] use a specifically constructed small subset of MSH WSD containing abbreviations with little ambiguity (average ambiguity 2.11) and artificially balanced (the average ratio of long-form examples to total examples for each short-form is 1:2). On this data set, UAD outperforms both SBE [63] and Distr. Sim. [22], see Table 7.2. Due to the small size and artificial balancing of MSH WSD, we do not consider it to be representative for abbreviation disambiguation, and therefore proceed to evaluate on larger unbiased data.

In Table 7.3 we show results of experimental comparisons on the Wikipedia and PubMed data sets. For both data sets, the FREQUENCY baseline manifests a discrepancy between the weighted and macro- measures. This is due to the disambiguation strategy which always disambiguates to the most popular long-form. Thus, it achieves perfect scores when disambiguating examples where the correct answer is the most popular long-form. Weighted performance is higher since less frequent
long-forms matter less in that measure. SBE achieves performance results similar to
what is reported in the original paper. Distr. Sim. is weaker than SBE in all mea-
sures. This is expected as Distr. Sim. is a variation of TBE, a weaker sibling of SBE,
both proposed at the same time by Li et al. [63]. Our model, UAD, outperforms the
baseline and both competitors in all measures, on both data sets.

Effect of background information
As described earlier, we expected both data sets to exhibit relatively high ratios be-
tween vocabulary and corpus length. Indeed, in Table 7.1, we can see that for the
Wikipedia data set, the ratio vocabulary size to corpus length is 1:73, while it is
1:266 for PubMed. Given the relatively high ratio of vocabulary to corpus length,
especially for the Wikipedia data set, word2vec might not see each word in the vo-
cabulary enough times in order to properly place each word in the vector space in
relation to its context.

We evaluated the benefit of augmenting with background knowledge in two ways:
directly and indirectly. The former consists of generating a background corpus of
sentences extracted from the same data set as the examples of abbreviation use. Thus,
it provides the same text style as the corpus of ambiguous abbreviation usage. The
indirect method aims to study whether performance can be improved without in-
creasing training time. For this, we used the GNEWS pre-computed vector space
[76]. The GNEWS space is trained on news items, so it does not share writing style
with either of our corpora, but we consider it highly relevant since it was trained on
a large corpus. In Table 7.1, we show the vocabulary overlap of the learning text
corpus (consisting of the examples extracted from Wikipedia and PubMed) with the
background text (denoted TXT), and with GNEWS.

Rows 2 of Tables 7.4 and 7.5 show that, indeed, addition of background text can
improve performance. For Wikipedia a small increase can be observed in macro-
recall indicating that background text improves performance on low-frequency ab-
breviations. For the other measures, we observe a small decrease in performance on
the Wikipedia data set. In the case of the PubMed data set, accuracy, weighted recall,
and macro-precision improve. Accuracy and weighted recall exhibit an improvement
of 6.7 points.

Following, we initialized the vector space to the values in the GNEWS pre-
computed vector space, and trained with our examples. Since GNEWS space is
based on a large collection of news items, we expected initialization with this vector
space to bring our own word vectors closer to the convergence point. Results for
this experiment are presented in row 3 of Tables 7.4 and 7.5 (compare row 1 vs. 3
in both tables). For Wikipedia, just as before, macro-recall is improved, suggest-
ing improved performance on low-frequency abbreviations. For PubMed, accuracy,
weighted recall, and macro precision improve, while weighted precision and macro-
recall are stable, or show little change. We believe the smaller improvements (and
the drops in performance) are partly due to the lower vocabulary overlap with the
### 7.4. EXPERIMENTS

#### Table 7.3: Baseline and competitor comparisons

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FREQUENCY</td>
<td>78.31</td>
<td>64.82</td>
<td>70.12</td>
<td>28.43</td>
<td>42.73</td>
<td>33.67</td>
</tr>
<tr>
<td>SBE [63]</td>
<td>89.54</td>
<td>93.40</td>
<td>89.79</td>
<td>84.92</td>
<td>88.43</td>
<td>85.41</td>
</tr>
<tr>
<td>Distr. Sim. [22]</td>
<td>87.84</td>
<td>92.36</td>
<td>89.31</td>
<td>82.39</td>
<td>86.55</td>
<td>83.00</td>
</tr>
<tr>
<td>UAD with TXT</td>
<td>94.28</td>
<td><strong>96.17</strong></td>
<td><strong>94.76</strong></td>
<td><strong>90.84</strong></td>
<td><strong>93.29</strong></td>
<td><strong>90.98</strong></td>
</tr>
</tbody>
</table>

#### Table 7.4: UAD performance on the Wikipedia data set. w2v trained with model=Skip-gram, w=10, i=10, size=300, neg=5

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Augmented Prec.</th>
<th>Augmented Rec.</th>
<th>Augmented F1</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>91.07</td>
<td>94.85</td>
<td><strong>92.01</strong></td>
<td>86.91</td>
<td>90.09</td>
<td>86.45</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT2</td>
<td>91.10</td>
<td><strong>91.10</strong></td>
<td>91.91</td>
<td>87.61</td>
<td>89.56</td>
<td><strong>86.45</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNEWS</td>
<td>91.76</td>
<td>94.90</td>
<td>92.44</td>
<td>88.00</td>
<td>89.35</td>
<td>86.75</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT + GNEWS</td>
<td>90.76</td>
<td>94.85</td>
<td>91.80</td>
<td>87.53</td>
<td>89.47</td>
<td>86.40</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 7.5: UAD performance on the PubMed data set. w2v trained with model=Skip-gram, w=10, i=10, size=300, neg=5

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Augmented Prec.</th>
<th>Augmented Rec.</th>
<th>Augmented F1</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>93.24</td>
<td>95.88</td>
<td>93.89</td>
<td>89.60</td>
<td>92.60</td>
<td>89.62</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT</td>
<td>93.06</td>
<td>96.43</td>
<td>94.16</td>
<td>90.81</td>
<td>93.26</td>
<td><strong>90.97</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GNEWS</td>
<td>93.26</td>
<td>93.92</td>
<td>93.89</td>
<td>89.57</td>
<td>92.61</td>
<td>89.61</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT + GNEWS</td>
<td><strong>94.37</strong></td>
<td><strong>96.15</strong></td>
<td><strong>94.37</strong></td>
<td><strong>90.81</strong></td>
<td><strong>93.27</strong></td>
<td><strong>90.97</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 7.6: UAD performance on Wikipedia without stop-words. w2v trained with model=Skip-gram, w=10, i=10, size=300, neg=5

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Augmented Prec.</th>
<th>Augmented Rec.</th>
<th>Augmented F1</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>78.03</td>
<td>92.41</td>
<td>75.29</td>
<td>75.17</td>
<td>75.17</td>
<td>75.17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT</td>
<td>77.62</td>
<td><strong>92.41</strong></td>
<td>75.05</td>
<td><strong>84.53</strong></td>
<td><strong>74.93</strong></td>
<td><strong>74.93</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 7.7: UAD performance on PubMed without stop-words. w2v trained with model=Skip-gram, w=10, i=10, size=300, neg=5

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Augmented Prec.</th>
<th>Augmented Rec.</th>
<th>Augmented F1</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>71.23</td>
<td>92.53</td>
<td>73.85</td>
<td>73.40</td>
<td>73.40</td>
<td>73.40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TXT</td>
<td>77.62</td>
<td><strong>92.41</strong></td>
<td>75.05</td>
<td><strong>84.53</strong></td>
<td><strong>74.93</strong></td>
<td><strong>74.93</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Table 7.8: Correlation between distance to closest other long-form and misclassification rate.

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Pearson Misclassification Rate</th>
<th>Spearman Misclassification Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wikipedia</td>
<td>−0.73</td>
<td>−0.43</td>
</tr>
<tr>
<td>PubMed</td>
<td>−0.57</td>
<td>−0.37</td>
</tr>
</tbody>
</table>
GNEWS vectors (24.85% and 3.4%) compared to the background text (54.10% and 34.47%), see Table 7.1.

Finally, rows 4 of Tables 7.4 and 7.5 contain results obtained when utilizing both kinds of augmentation. For both data sets, this leads to mixed results with the disambiguator benefiting more on the PubMed data set. In fact, comparing rows 2, 3, and 4 for PubMed, we can conclude that augmentation with background text is more effective than augmentation with GNEWS. The reverse is true for Wikipedia where seeding the vector space with GNEWS outperforms augmentation using background text. This might be due to the higher similarity of newswire text style to encyclopedic text than to scientific text.

The results of these experiments reveal an important aspect of our proposed unsupervised method: disambiguation performance can be improved by adding word usage information. Both augmentations are easy to employ as more vector spaces pre-computed over large corpora are becoming available.

**Effect of stop-word removal**

To evaluate the impact of stop-words, we re-processed our data sets and removed stop-words. A list of the eliminated tokens is available together with the Wikipedia data set and the evaluation script at http://bit.ly/2DTkMh7.

In Tables 7.6 and 7.7 we show the results of disambiguation on the two data sets after stop-word removal. For both data sets, the non-augmented vector spaces lead to disambiguation accuracy, precision, and recall that are higher than the augmented ones for spaces that contain stop-words. This is expected, given the way word2vec’s underlying language model works, since the removal of stop-words eliminates noise from the window around each long-form. Results also show that it is highly important that the vector for each long-form is constructed only out of those words in the context that are semantically relevant. Furthermore, augmentation with background text or pre-trained vectors can still be applied and continue.

**Comparison with Continuous Bag of Words**

word2vec supports two models for learning of word vectors: the Continuous Bag of Words Model (CBOW) and Skip-gram. In Tables 7.9 to 7.12 we provide experiments using the CBOW model instead of Skip-gram. We use the same setup as our Skip-gram experiments in Tables 7.4 to 7.7.

UAD performs better when using the Skip-gram model compared to CBOW on both of our data sets. We believe this is due to the learning strategy in CBOW where a context vector is constructed and, from the context vector, a prediction is made. During training, the neural network assigns the prediction error to the context vector. Since it is not possible to determine which member of the context window is responsible for the error, the same correction is applied to the vectors of all words in the window. The Skip-gram model uses pairs of words and thus, vector corrections are applied proportionately to prediction error of each word’s vector. Our experimental
observation is in line with the original work of Mikolov et al. [75], who also conclude that word2vec with the Skip-gram model leads to more qualitative vector spaces.

However, some tendencies can be observed between our CBOW and Skip-gram experiments. First, addition of background data in the form of text or initialization with pre-trained vectors tends to improve disambiguation quality showing the importance of using more text to derive word vectors (e.g., row 2 in Table 7.9). However, augmentation text from the same corpus generally leads to the highest disambiguation performance, probably due to matching text styles (e.g., in Table 7.11 row 2 has a higher performance increase than row 3). Secondly, we observe that just as with Skip-gram, UAD based on CBOW performs worse on the PubMed data set, probably due to the more complex language (e.g., row 2 in Table 7.12 shows weaker disambiguation performance than row 2 in Table 7.11).

**Comparisons between word2vec and GloVe**

As discussed in Section 7.3, GloVe [96] is a method for unsupervised construction of word vectors from large text corpora which attempts to combine word2vec’s Skip-gram analogy capabilities with global corpus co-occurrence information. It is often considered an alternative to word2vec, especially for Named Entity Recognition, or tasks that involve similarity.

In Table 7.13, we show a version of Table 7.3 with results from disambiguation with UAD, but using GloVe as a drop-in replacement for word2vec for construction of word vectors (see line 4). As mentioned earlier, we expect GloVe vectors to be less suited for abbreviation disambiguation, as the local context is of decisive importance for the representation of long-forms in the vector space.

As expected, the experiments show that UAD using GloVe performs significantly worse than UAD using word2vec vectors. This observation is also in line with Charbonnier and Wartena [22], who also tested GloVe as part of Distr. Sim. and noticed a performance decrease. In fact, UAD with GloVe only outperforms the baseline FREQUENCY disambiguator. The results suggest that even though word2vec and GloVe are drop-in replacements for one-another in tasks that rely heavily on word similarity, that is not the case for abbreviation disambiguation, where local context is of high relevance.

**Evaluation against human-labeled data**

Both our learning and testing data are automatically extracted using the pipeline described in Section 7.3. To evaluate the reliability of results on these data sets, we manually labeled 7000 examples from one of the 10 folds from our Wikipedia data set. We then trained UAD using only the other 9 folds under the Skip-gram model with background text augmentation.

On the manually labeled subset, UAD achieves a weighted precision and recall of 97.62 and 95.18, respectively; macro-precision and recall of 93.24 and 94.09.
### Table 7.9: UAD performance on the Wikipedia data set. \textit{w2v} trained with 
\textit{model=CBOW, w=10, i=10, size=300, neg=5}

<table>
<thead>
<tr>
<th>Augmented</th>
<th>Weighted Acc.</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 No</td>
<td>77.91</td>
<td>92.30</td>
<td>77.91</td>
<td>80.42</td>
<td>81.23</td>
<td>85.24</td>
<td>80.24</td>
</tr>
<tr>
<td>2 No2</td>
<td>76.72</td>
<td>92.13</td>
<td>76.72</td>
<td>79.36</td>
<td>80.69</td>
<td>84.89</td>
<td>79.53</td>
</tr>
<tr>
<td>3 TXT</td>
<td>79.31</td>
<td>92.47</td>
<td>79.31</td>
<td>82.25</td>
<td>82.00</td>
<td>85.49</td>
<td>80.94</td>
</tr>
<tr>
<td>4 GNEWS</td>
<td>78.05</td>
<td>92.33</td>
<td>78.05</td>
<td>80.70</td>
<td>81.31</td>
<td>85.29</td>
<td>78.05</td>
</tr>
<tr>
<td>5 TXT + GNEWS</td>
<td>79.46</td>
<td>92.51</td>
<td>79.46</td>
<td>82.37</td>
<td>81.99</td>
<td>85.56</td>
<td>80.99</td>
</tr>
</tbody>
</table>

### Table 7.10: UAD performance on the PubMed data set. \textit{w2v} trained with 
\textit{model=CBOW, w=10, i=10, size=300, neg=5}

<table>
<thead>
<tr>
<th>Augmented</th>
<th>Weighted Acc.</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 No</td>
<td>69.31</td>
<td>86.82</td>
<td>69.31</td>
<td>74.18</td>
<td>61.69</td>
<td>69.31</td>
<td>60.24</td>
</tr>
<tr>
<td>2 TXT</td>
<td>69.78</td>
<td>86.89</td>
<td>69.78</td>
<td>74.60</td>
<td>61.77</td>
<td>69.24</td>
<td>60.30</td>
</tr>
<tr>
<td>3 GNEWS</td>
<td>68.99</td>
<td>86.85</td>
<td>68.99</td>
<td>73.94</td>
<td>61.62</td>
<td>69.32</td>
<td>60.15</td>
</tr>
<tr>
<td>4 TXT + GNEWS</td>
<td>69.94</td>
<td>86.89</td>
<td>69.94</td>
<td>74.75</td>
<td>61.78</td>
<td>69.40</td>
<td>60.43</td>
</tr>
</tbody>
</table>

### Table 7.11: UAD performance on the Wikipedia data set without stop-words. \textit{w2v} trained with 
\textit{model=CBOW, w=10, i=10, size=300, neg=5}

<table>
<thead>
<tr>
<th>Augmented</th>
<th>Weighted Acc.</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 No</td>
<td>74.69</td>
<td>88.94</td>
<td>74.69</td>
<td>78.94</td>
<td>67.31</td>
<td>77.37</td>
<td>68.06</td>
</tr>
<tr>
<td>2 TXT</td>
<td>75.74</td>
<td>88.89</td>
<td>75.74</td>
<td>79.66</td>
<td>67.78</td>
<td>77.14</td>
<td>68.51</td>
</tr>
<tr>
<td>3 GNEWS</td>
<td>73.33</td>
<td>88.92</td>
<td>73.33</td>
<td>77.87</td>
<td>66.99</td>
<td>77.34</td>
<td>67.69</td>
</tr>
<tr>
<td>4 TXT + GNEWS</td>
<td>75.58</td>
<td>88.91</td>
<td>75.58</td>
<td>79.58</td>
<td>67.74</td>
<td>77.19</td>
<td>68.48</td>
</tr>
</tbody>
</table>

### Table 7.12: UAD performance on the PubMed data set without stop-words. \textit{w2v} trained with 
\textit{model=CBOW, w=10, i=10, size=300, neg=5}

<table>
<thead>
<tr>
<th>Augmented</th>
<th>Weighted Acc.</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 No</td>
<td>74.69</td>
<td>88.94</td>
<td>74.69</td>
<td>78.94</td>
<td>67.31</td>
<td>77.37</td>
<td>68.06</td>
</tr>
<tr>
<td>2 TXT</td>
<td>75.74</td>
<td>88.89</td>
<td>75.74</td>
<td>79.66</td>
<td>67.78</td>
<td>77.14</td>
<td>68.51</td>
</tr>
<tr>
<td>3 GNEWS</td>
<td>73.33</td>
<td>88.92</td>
<td>73.33</td>
<td>77.87</td>
<td>66.99</td>
<td>77.34</td>
<td>67.69</td>
</tr>
<tr>
<td>4 TXT + GNEWS</td>
<td>75.58</td>
<td>88.91</td>
<td>75.58</td>
<td>79.58</td>
<td>67.74</td>
<td>77.19</td>
<td>68.48</td>
</tr>
</tbody>
</table>
7.4. EXPERIMENTS

Wikipedia (no stop-words)

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Weighted Acc.</th>
<th>Weighted Prec.</th>
<th>Weighted Rec.</th>
<th>Weighted F1</th>
<th>Macro Acc.</th>
<th>Macro Prec.</th>
<th>Macro Rec.</th>
<th>Macro F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 FREQUENCY</td>
<td>78.31</td>
<td>64.82</td>
<td>78.31</td>
<td>70.12</td>
<td>28.43</td>
<td>42.73</td>
<td>33.67</td>
<td></td>
</tr>
<tr>
<td>2 SBE [63]</td>
<td>89.54</td>
<td>93.40</td>
<td>89.54</td>
<td>90.79</td>
<td>84.92</td>
<td>88.43</td>
<td>85.41</td>
<td></td>
</tr>
<tr>
<td>3 Distr. Sim. [22]</td>
<td>87.84</td>
<td>92.36</td>
<td>87.84</td>
<td>89.31</td>
<td>82.39</td>
<td>86.55</td>
<td>83.00</td>
<td></td>
</tr>
<tr>
<td>4 UAD using GloVe</td>
<td>82.23</td>
<td>89.43</td>
<td>82.23</td>
<td>82.90</td>
<td>74.50</td>
<td>74.05</td>
<td>68.99</td>
<td></td>
</tr>
<tr>
<td>5 UAD with TXT</td>
<td><strong>94.28</strong></td>
<td><strong>96.17</strong></td>
<td><strong>94.28</strong></td>
<td><strong>94.76</strong></td>
<td><strong>90.84</strong></td>
<td><strong>93.29</strong></td>
<td><strong>90.98</strong></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.13: Comparisons using GloVe on the Wikipedia data set with no stop-words.

These results are close to the ones in Table 7.6, which indicates that the automatically extracted data set is of high quality and confirms the results we presented earlier.

Since the labeled data set is much smaller than the ones automatically created, it better lends itself to detailed error analysis. We used 5 categories to classify each disambiguation error. Of the 337 errors, 2.33% are due to multi-level abbreviations (e.g., *Communist_Party_of_the_United_States* vs. *Communist_Party_USA*). 9.67% are due to inconsistencies in long-forms that our normalization step cannot handle (e.g., *Average Annual Daily Traffic* vs. *Annual Average Daily Traffic*). A portion of 5.33% represents language mismatches in long-forms that mean the same thing (e.g., *Federation of Association Football* vs. *Fédération Internationale de Football Association*). The second-largest source of error is due to incorrect long-form mappings in our pipeline (e.g., *Advanced Placement Program* instead of *Advanced Placement*). We believe most of these errors can be solved in the future through improved pre-processing.

Of the remaining mistakes, many are made for long-forms that appear in sufficiently different contexts while only a minority represent difficult cases that have similar contexts such as *American Broadcasting Company* versus *Australian Broadcasting Corporation*.

Pre-evaluation analysis

In order to demonstrate that UAD supports efficient and cost-effective pre-evaluation analysis, we investigate the correlation between cosine distance of long-form pairs and their misclassification rate. For each of the data sets, we selected the models corresponding to row 2 in Tables 7.6 and 7.7. For each pair of long-forms, we calculated the cosine distance and the misclassification rate between the two long-forms, and then both their Pearson and Spearman correlation coefficients. The *Pearson Correlation Coefficient* $\rho_P$ is defined as:

$$\rho_P = \frac{\text{cov}(X,Y)}{\sigma_x \sigma_y}$$

(7.5)
While the Spearman Correlation Coefficient $\rho_S$ is defined as:

$$\rho_S = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$  \hspace{1cm} (7.6)

While Pearson evaluates the linear relationship between two variables and assumes normally distributed data, Spearman evaluates the monotonic relationship. For example, if an increase in one variable leads to an increase in the other, but the relationship is not linear, the Pearson correlation coefficient will show a weaker correlation, while Spearman will not be affected by the lack of linearity. We, therefore, measure both correlation coefficients in order to obtain a clearer picture. Analysis results are shown in Table 7.8.

Both Pearson and Spearman correlation coefficients show that, on both data sets, there is a strong negative correlation between cosine distance and misclassification rate. In other words, the closer two long-forms are to each other, the more likely it is that UAD will have difficulties selecting the correct disambiguation. This follows the hypothesis presented in Section 7.3: if two long-forms have similar representatives, then a context vector aligned with one of them, will also align with the other.

Pairs of close long-forms provide information as to how disambiguation performance can be improved. More specifically cases where long-forms represent the same meaning, but are considered different due to: lack of support for multi-level abbreviations (e.g., United States Geological Survey and U.S. Geological Survey) for USGS, language mismatches, difficult edge cases for long-form normalization (e.g., words swapped around like in House Committee on Un-American Activities and House Un-American Activities Committee for HUAC). Finally, we also observe examples from cases where more data is required for proper disambiguation due to long-forms that are difficult to disambiguate due to them denoting concepts in the same domains (e.g., Metropolitan Railway and Midland Railway or American Broadcasting Company and Australian Broadcasting Corporation).

The pre-evaluation analysis feature of UAD is extremely useful in practice as it allows the identification of abbreviations that are potentially difficult to disambiguate without requiring expensive evaluations, such as the 10-fold cross-validation we performed in this article. Abbreviations that are difficult to disambiguate can either be removed from the models, investigated for potential pre-processing errors or noise. They can be used to target corpus acquisition towards gathering more usage examples for the specific abbreviations, which can lead to better representations for the long-forms and, thus, higher disambiguation performance.

Finally, for pairs of long-forms that denote the same meaning, the model can either be retrained after normalizing the long-forms to a single lexical representation, or the model can be updated directly by replacing the two long-form vectors with one that represents their average, thus completely avoiding spending time on model retraining. In Ciosici and Assent [29] we present a system that can achieve these tasks and evaluate the performance of various model correction strategies for UAD.
7.5 Conclusion

We presented Unsupervised Abbreviation Disambiguation (UAD), a fully unsupervised method for abbreviation disambiguation that does not require hand-designed features or any labeled data. UAD automatically identifies abbreviations used in a large corpus of unstructured text, determines the potential long-form meanings, and constructs a model that can disambiguate abbreviations based on context.

UAD creates word vector spaces for representation of long-forms relative to their context, by introducing distinct tokens for each long-form identified. The relation between long-form representations and the context surrounding ambiguous abbreviations thus captures the information required for successful disambiguation.

Through a thorough empirical evaluation, we demonstrate that UAD outperforms a realistic baseline and state-of-the-art methods. Our evaluation is based on two data sets that are at least one order of magnitude larger than previously used in literature, are more ambiguous, have not been artificially balanced, and are more representative of real-world performance.

We also presented methods to further improve UAD’s performance through augmentation with easy to acquire, ubiquitous background knowledge in the form of unstructured text, or pre-computed vector spaces.

This the first method that supports insights into disambiguation performance without requiring full-scale evaluation through pre-evaluation analysis which can help identify problematic abbreviations, help target corpus acquisition, and in some cases allow for model adjustments that do not require retraining.

Both augmentation and pre-evaluation analysis make UAD highly relevant for real-world use, especially in domains with thousands of ambiguous abbreviations, or those that lack large sets of manually annotated data.

As our contribution to the research community, we publish a data set containing abbreviation annotations, which is at least an order of magnitude larger than current similar resources. We hope the data set will support repeatability and further research in abbreviation disambiguation.
Chapter 8

Abbreviation Expander - A web-based system for easy reading of technical documents

Abstract

Abbreviations and acronyms are a part of textual communication in most domains. However, abbreviations are not necessarily defined in documents that employ them. Understanding all abbreviations used in a given document often requires extensive knowledge of the target domain and the ability to disambiguate based on context. This creates considerable entry barriers to newcomers and difficulties in automated document processing. Existing abbreviation expansion systems or tools require substantial technical knowledge for set up or make strong assumptions which limit their use in practice. Here, we present Abbreviation Expander, a system that builds on state of the art methods for identification of abbreviations, acronyms and their definitions and a novel disambiguator for abbreviation expansion in an easily accessible web-based solution.

8.1 Introduction

Abbreviations and acronyms are often used in text documents and denote typically long, often domain-specific, concepts that authors need to refer to multiple times. However, the use of abbreviations and acronyms can make reading and understanding difficult for people new to a specific field, can lead to confusion, and make automated text processing challenging, for example, in indexing text documents. Unfortunately, expanding abbreviations is a complex task. The meaning of some abbreviations and acronyms (e.g. DNA meaning deoxyribonucleic acid in biology-related domains) is often considered well-known, and is rarely defined in documents using them. Other
abbreviations and acronyms can denote multiple concepts, depending on their context (e.g. PCB can refer to a number of distinct concepts\(^1\)).

Available abbreviation expansion systems are limited in their usefulness either due to requiring technical knowledge on the user side or by relying on simple, dictionary-based methods which cannot be applied to ambiguous abbreviations that have more than one meaning. We present a system that automatically expands abbreviations and acronyms in a user provided document. Our system is a web-based application that does not require that users have experience setting up pipelines for Natural Language Processing. We build on state-of-the-art Natural Language Processing techniques and a novel disambiguation method based on unsupervised learning.

### 8.2 System Architecture

By building a web application, we aim to make users oblivious to the technical complexities of processing natural language. From a user’s point of view, they upload a text file to the system and immediately see the file’s content with all abbreviations and acronyms expanded.

Figure 8.1 shows the architecture of Abbreviation Expander’s back-end. Text is first tokenized and split into sentences, after which a number of abbreviation expanders are used. Finally, their results are combined. The biggest part of our system is composed of the processing pipeline. We use the UIMA\(^2\) framework as the basis for our system because it provides mature support for construction of processing pipelines and benefits from a wide-array of external NLP libraries. The wide support for libraries allows us to employ established tools for pre-processing steps such as tokenization and sentence splitting for which we use the Stanford Core NLP library.

---

\(^1\)https://en.wikipedia.org/wiki/PCB

\(^2\)https://uima.apache.org
We separate the abbreviation and acronym expansion into four different components: three components that perform expansion and one that combines their outputs in order to achieve consistency.

Before we describe the various components in detail, we establish some definitions. An **unambiguous abbreviation or acronym** is one that never expands into more than a single long-form. This corresponds to a one-to-one mapping. For example, in all of English Wikipedia we could only find one meaning for the acronym SS-RMS, meaning *Space Station Remote Manipulator System*, popularly referred to as *Canadarm2*. An **ambiguous abbreviation or acronym** is one that can expand to multiple long-forms and the correct expansion is dependent on the context. However, *an unambiguous use* of an ambiguous abbreviation or acronym is one where an ambiguous abbreviation or acronym is used in such a way that the correct expansion is obvious. One such case is the definition of an abbreviation, such as *The Mobile Servicing System (MSS)*, is a robotic system on board the *International Space Station*. Because of the definition, it is clear which expansion is intended by the author.

The **Pattern Annotator** component is a re-implementation of the rules presented by [106]. It uses linguistic patterns to identify definitions of abbreviations and acronyms. More specifically, it looks for either the pattern `<short-form>(text)`, or the reverse `text(<short-form>)`. For each identified instance, it attempts to find a long-form expansion in the text preceding the parenthesis or contained in the parenthesis, respectively. This component can thus identify abbreviations and acronyms defined directly in the user-provided text. Our implementation differs from that of BADREX [47] by the fact that it more closely follows the extraction rules defined in [106]. At the same time, we provide added support for various edge cases. For example, the original method does not support mapping of long-forms to short-forms when the long form contains two words at the beginning that start with the same letter (for example, OAS meaning *Organization of American States* is wrongly mapped to *of American States*).

Our system solves this by a combination of looking ahead and a small set of stopwords not to be considered for the first word in a long-form (e.g. which, where, at, on, ...).

The **Dictionary Annotator** component uses a dictionary of unambiguous abbreviations and acronyms, that we automatically extracted from English Wikipedia. We pre-processed English Wikipedia using only our Pattern Annotator in order to extract all abbreviations and acronyms that are unambiguous. Unambiguous abbreviations and acronyms have a one-to-one mapping between long-forms and short-forms. This annotator gives our system the ability to expand abbreviations and acronyms that are not defined in the text, but are known to only ever mean one thing. By focusing exclusively on unambiguous abbreviations and acronyms, this annotator avoids the pitfalls of dictionary based systems described in Section 8.3, i.e., the annotator avoids creating wrong expansions for abbreviations which can mean multiple things by working exclusively with abbreviations known to be unambiguous.

The third component that performs expansions, the **Vector Space Annotator** deals...
exclusively with ambiguous abbreviations and acronyms. It uses the context surrounding a short-form and a pre-computed vector space in order to disambiguate the abbreviation. The vector space is based on sentences from English Wikipedia containing ambiguous abbreviations (meaning abbreviations containing a one-to-many mapping between short-forms and long-forms) that are used in an unambiguous way (meaning that we already know which one of the multiple expansions is the correct one). We extracted these sentences using our Pattern Annotator. The Vector Space Annotator can thus expand abbreviations and acronyms that can have multiple meanings and whose definitions do not appear in the user provided text.

Finally, the Expansion Combiner uses the annotations from the previous components and combines them into consistent overall expansions. Please note that it is possible that two annotators expand an abbreviation to different long-forms. For example, the user provided text might introduce a new meaning for an abbreviation that we know as unambiguous and so, the Pattern Annotator and Dictionary Annotator might disagree. Similarly, the Pattern Annotator and Vector Space Annotator might arrive at different expansions if the author uses a new meaning for a known ambiguous abbreviation, or if the Vector Space Annotator should output an incorrect expansion. Finally, since the Vector Space Annotator works on one sentence at a time, it is possible that it disambiguates the same abbreviation to different long-forms in different sentences, thus leading to inconsistencies. The Expansion Combiner addresses these cases by implementing a priority system and, for the Vector Space Annotator specifically, a voting system.

Figure 8.2 shows a screenshot from the Abbreviation Expander system. In the example text, the original definition given in the text is marked and all subsequent uses of the short-form are preceded by the abbreviation’s expansion. Users can verify how the system arrived at a specific expansion by clicking on the inserted expanded form, see Figure 8.3. The system features a menu where users can input or open some pre-loaded text files.
8.3. RELATED WORK

BADREX [47] is a plugin for the GATE [34] text analysis framework. It performs abbreviation expansion using dynamic regular expressions based on linguistic patterns for their definition [106]. The system requires an installation of GATE and familiarity with establishing GATE pipelines and loading plugins. It can identify abbreviation and acronym definitions in text and can then co-reference other instances of the identified short-forms to the found definition. BADREX cannot perform abbreviation disambiguation, i.e., it cannot handle ambiguous cases as it relies exclusively on the definitions present in the document.

Web browser-based systems, like ABBREX [1], can be installed in a browser and expand abbreviations found on web pages. The expansion is based on stored dictionaries of abbreviations and lists of web pages they apply to. Thus, they cannot pick up definitions in text, or perform disambiguation. Being a dictionary-based expander, ABBREX assumes a one-to-one mapping between abbreviations and their long-forms, which means that in the case of ambiguous abbreviations, it has no other alternative, but to expand to whichever long-form is stored in the dictionary.

Another type of system, found e.g. in commercial software [9, 111], tries to expand user-defined abbreviations at writing time. This kind of software targets a different use case and cannot be applied to already written text.

The problem of matching abbreviations and acronyms with their long-forms has also been studied in research such as [80, 124]. However, they assume supervised learning settings, where a large amount of human effort has to go into providing ground truth examples. Also, they generally focus on methods, and do not provide (online) systems that users can easily use.

Abbreviation Expander presents a working web-based solution that does not require supervised ground truth information, and that can handle both unambiguous and ambiguous cases.

8.4 Conclusion

We present Abbreviation Expander, a web-based system that allows users to expand abbreviations and acronyms in text documents. Our system builds on state of the
art methods for identification of abbreviations, acronyms and their definitions and our novel disambiguator based on word vector spaces. The Vector Space Annotator is still under active research and will be described in detail in a research paper in the near future. Abbreviation Expander requires no technical knowledge on part of its users and reliably expands both unambiguous and ambiguous abbreviations, improving text understanding and access in practice. In the future we plan to include feedback features into the system so that users can reject wrong expansions.
Chapter 9

Abbreviation Explorer - an interactive system for pre-evaluation of Unsupervised Abbreviation Disambiguation

Abstract

We present Abbreviation Explorer, a system that supports interactive exploration of abbreviations that are challenging for Unsupervised Abbreviation Disambiguation (UAD). Abbreviation Explorer helps to identify long-forms that are easily confused, and to pinpoint likely causes such as limitations of normalization, language switching, or inconsistent typing. It can also support determining which long-forms would benefit from additional input text for unsupervised abbreviation disambiguation. The system provides options for creating corrective rules that merge redundant long-forms with identical meaning. The identified rules can be easily applied to the already existing vector spaces used by UAD to improve disambiguation performance, while also avoiding the cost of retraining.

9.1 Introduction

Abbreviations are short forms of concepts that authors employ to avoid repeated typing of long text sequences (e.g. National Aeronautics and Space Administration is abbreviated to NASA). Because long-forms are represented by shorter sequences of characters (often just two or three), even in the same domain, multiple concepts may map to the same short-form. Such ambiguous abbreviations, i.e., short-forms with several potential meanings, are quite common. A study by Liu et al. [68] showed that 23.1% of abbreviations in the Unified Medical Language System (UMLS) ontology, a popular resource in the field of medicine, are ambiguous, meaning that they map to more than one long-form.
Abbreviation disambiguation is the task of identifying the intended long-form for an ambiguous short-form, given its use in a sentence. Expansion and disambiguation of abbreviations can make technical text easier to read [30]. The task of disambiguating abbreviations in context has been included in both the 2013 and 2014 ShAReCLEF eHealth Challenge [84].

Unsupervised Abbreviation Disambiguation (UAD) [26] is a recent unsupervised approach that identifies ambiguous abbreviations, and makes use of word embeddings to identify the intended long-forms given contextual uses of ambiguous abbreviations. While successfully disambiguating the vast majority of the ambiguous abbreviations, some difficult cases remain. Following an idea of pre-evaluation in suggested by Ciosici et al. [26], we present a system called Abbreviation Explorer that supports investigation and correction of word embedding spaces learned by UAD. Highlighting difficult cases, it allows inspection of likely causes and potential resolution through rewrite rules. The rules can either be used to retrain UAD, or directly applied to its vector space, thus eliminating the need to retrain the model. Abbreviation Explorer does not target simple variations in long-forms such as plurals, hyphenation, or minor text variations as identified by Moon et al. [80], Zhou et al. [126]. Such noise is eliminated by UAD’s pre-processing pipeline. Rather, it focuses on semantically challenging abbreviations, or those where the unstructured text input corpus provided to UAD does not provide sufficient context for successful disambiguation. Abbreviation Explorer is a general tool for long-form normalization that does not require expert knowledge or domain-specific, human-curated knowledge bases that approaches like Melton et al. [73] rely on.

Abbreviation Explorer thus supports the user in understanding which abbreviations are difficult to disambiguate, why UAD might not have been able to learn how they differ, in issuing corrections that immediately improve the disambiguation performance, or in determining where more input text would allow learning of better representations and result in improved disambiguation. All this, without having to conduct expensive, large-scale evaluation on abbreviation disambiguation tasks that require manually labeled data. In fact, Abbreviation Explorer takes as input only the vector model trained by UAD and the training data that was used to generate the vector model. 1

9.2 Pre-evaluation analysis

Long-forms that map into the same short-form and are close to each other in the UAD vector space are correlated with low disambiguation performance [26]. Long-forms end up close to each other in the vector space due to incorrect long-form normalization in UAD, inconsistent reference by humans (annual average daily traffic vs. average annual daily traffic), language switching (Centre National de la Recherche Scientifique vs. Centre for Scientific Research), multi-layer abbreviations (Voice over IP vs. Voice over Internet Protocol), organization name changes (Organisation of Is-
9.2. PRE-EVALUATION ANALYSIS

Figure 9.1: The main screen of Abbreviation Explorer listing long-forms that are challenging for disambiguation.

Figure 9.2: Detail screen with subset of words that appear often in the context of the long-forms average annual daily traffic and annual average daily traffic. The large frequency of contextual terms indicates that the two long-forms likely denote the same concept.

Islamic Cooperation vs. Organisation of the Islamic Conference), or lack of sufficient examples in the input text for a proper representation.

Abbreviation Explorer identifies these cases using the cosine distance between pairs of long-forms belonging to the same short-form, and presents them to the user for interactive exploration. Figure 9.1 shows the main screen of Abbreviation Explorer displaying a list of long-forms and the cosine distance between them. In some cases, it is obvious even without domain knowledge that the pairs are lexical variations that should be collapsed into a single long-form.

For less obvious cases, Abbreviation Explorer provides the user with a page containing information that supports investigation. It provides two indicators for contextual analysis: a list of most common words observed in training examples for both
Figure 9.3: Detail screen with subset of words close to the two challenging long-forms. Red rows indicate words that are close to both long-forms. The large overlap indicates that the two long-forms likely denote the same concept.

long-forms, and a list of the top 30 words in the word embedding space that are close to each long-form. The list of common words observed in training data for both long-forms helps identify cases where the two long-forms denote the same concept, or where the training data is inconclusive for effective disambiguation. The list of top words in the vector space can be used to pinpoint long-forms that denote the same concept even when the training data does not contain significant word overlap. Figures 9.2 and 9.3 show the two views for the abbreviation AADT which is expressed in the data as annual average daily traffic and average annual daily traffic. The view helps users conclude that the two long-forms are term variations of the same concept as they are used in similar contexts. On the other hand, long-forms that denote separate concepts often have little, if any, overlap between the sets of close words. In Figure 9.4, we can see that for the abbreviation ABC, its two correctly identified long-forms American Broadcasting Company and Australian Broadcasting Corporation have no shared context as they denote separate concepts.

Abbreviation Explorer supports two kinds of actions for pairs of close long-forms that should be corrected. The user can either choose to issue a manual rewrite rule, forcing the system to collapse one long-form into the other, or ask for a long-form to be deleted from the system. The latter action is useful for addressing issues in the input text processing that are not captured in UAD’s pre-processing. For example, in the case of the short-form FN, Front is picked up as a long-form instead of Front Nationale (which is the correct long-form) due to the greedy nature of UAD’s normalization which identifies Front as a long-form because it contains both letters F and N.

Rewrite and deletion rules from Abbreviation Explorer can be exported as a JSON document which can then be used in two ways. It can either be applied to the training
9.3 Evaluation of Abbreviation Explorer

In order to study the effect of manual corrections created using Abbreviation Explorer, we used the same Wikipedia data set that was used in the evaluation of UAD [26]. For all experiments, we employed 10-fold cross-validation, with the same folds used in the UAD evaluation. In experiments requiring UAD training, we used the same word2vec hyper-parameters as in UAD evaluation.

Table 9.1: Comparisons with UAD on data set Wikipedia (no-stopwords).

<table>
<thead>
<tr>
<th>Disambiguator</th>
<th>Acc.</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 UAD with TXT</td>
<td>94.28</td>
<td>96.17</td>
<td>94.28</td>
<td>94.76</td>
<td>90.84</td>
<td>93.29</td>
<td>90.98</td>
</tr>
<tr>
<td>2 Removed long-forms</td>
<td>96.38</td>
<td>97.34</td>
<td>96.38</td>
<td>96.61</td>
<td>92.72</td>
<td>95.15</td>
<td>93.15</td>
</tr>
<tr>
<td>3 Averaged long-form vectors</td>
<td>96.37</td>
<td>97.32</td>
<td>96.37</td>
<td>96.60</td>
<td>92.64</td>
<td>95.12</td>
<td>93.11</td>
</tr>
<tr>
<td>4 Retrained UAD with TXT</td>
<td>96.28</td>
<td>97.33</td>
<td>96.28</td>
<td>96.54</td>
<td>92.64</td>
<td>95.13</td>
<td>93.07</td>
</tr>
</tbody>
</table>

Figure 9.4: Detail screen with subset of words in the vector space close to the two challenging long-forms. Red rows indicate words that are close to both long-forms. The lack of vocabulary overlap indicates that the two long-forms denote separate concepts.
With Abbreviation Explorer, we identified 40 pairs of long-forms that denote the same concept. As mentioned earlier, correction rules can either be applied to the data followed by a retraining of UAD, or they can be applied directly to already trained UAD models. We study two ways to correct preexisting vector spaces: (1) for every pair of long-forms denoting the same concept remove one long-form from the vector space and (2) for every pair of long-forms denoting the same concept, we replace the two long-form vectors with their average. In Table 9.1, we provide a comparison of disambiguation performance following the three corrective methods. The first row contains disambiguation performance of UAD without any corrections; rows 2 and 3 contain the two vector adjustment methods applied to the vector space from the first row using corrections identified with Abbreviation Explorer; and the last row contains the performance of UAD retrained on data where we applied corrections identified with Abbreviation Explorer.

All three corrective methods outperform UAD on the original data, using no corrections, by up to 1.85 weighted F1 points. Both precision and recall are improved after applying the corrective rules. This illustrates the usefulness of Abbreviation Explorer in supporting users to issue manual corrections for difficult long-forms. Another important result in Table 9.1, is that the three correction methods studied result in performances within 0.07 weighted F1 points of each other. Removing a long-form from the vector space, or averaging two long-forms is considerably less costly than a complete re-derivation of the word embedding space utilized by UAD. This leads us to conclude that adjusting UAD’s vector spaces is preferred as it results in similar performance to retraining UAD, but does not incur the time cost of re-deriving word vector spaces.

9.4 Conclusion

We present Abbreviation Explorer, a system that supports interactive exploration of abbreviations that are easily confused by Unsupervised Abbreviation Disambiguation (UAD). Abbreviation Explorer works by identifying and understanding long-forms whose learned word embedding representations are close to each other and providing contextual information to help users understand which long-forms denote the same concept. Abbreviation Explorer can assist users who are not domain-experts in generating corrective rules that address abbreviation disambiguation difficulties due to incorrect long-form normalization, inconsistent typing, language switching, multi-layer abbreviations, organization name changes, and more. Corrective rules generated using Abbreviation Explorer can be used to improve the disambiguation performance of UAD, even without performing expensive full-scale evaluation using labeled data. The corrective rules can be applied to the already existing vector spaces used by UAD to improve disambiguation performance, while, at the same time, avoiding the cost of retraining UAD.
Bibliography


97


[23] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. 21


[34] Hamish Cunningham, Valentin Tablan, Angus Roberts, and Kalina Bontcheva. Getting more out of biomedical documents with gate’s full lifecycle open source text analytics.


[59] Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, et al. Universal dependencies 2.1. 28, 43, 57

[60] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. 10:707. 67


[63] Chao Li, Lei Ji, and Jun Yan. Acronym disambiguation using word embedding. In AAAI, pages 4178–4179. 32, 33, 36, 62, 63, 64, 73, 75, 76, 77, 81


[71] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In Association for Computational Linguistics (ACL) System Demonstrations, pages 55–60. URL http://www.aclweb.org/anthology/P/P14/P14-5010. 87


[79] Sungrim Moon, Bjoern-Toby Berster, Hua Xu, and Trevor Cohen. Word sense disambiguation of clinical abbreviations with hyperdimensional computing. In
BIBLIOGRAPHY

AMIA Annual Symposium Proceedings, volume 2013, page 1007. American Medical Informatics Association. 32, 33, 34, 62, 63, 71


[93] Naoaki Okazaki, Sophia Ananiadou, and Jun’ichi Tsujii. Building a high-quality sense inventory for improved abbreviation disambiguation. 26(9):1246–1253. 64, 68


[95] Olutobi Owoputi, Brendan O’Connor, Chris Dyer, Kevin Gimpel, and Nathan Schneider. Part-of-speech tagging for twitter: Word clusters and other advances. . 53


[106] A Schwartz and M Hearst. A simple algorithm for identifying abbreviation definitions in biomedical text. 8. 66, 87, 89


[116] Matthew C Swain and Jacqueline Manina Cole. ChemDataExtractor: A toolkit for automated extraction of chemical information from the scientific literature. page acs.jcim.6b00207. ISSN 1549-9596. doi: 10.1021/acs.jcim.6b00207. URL http://pubs.acs.org/doi/abs/10.1021/acs.jcim.6b00207. 18, 27, 28, 40, 41


