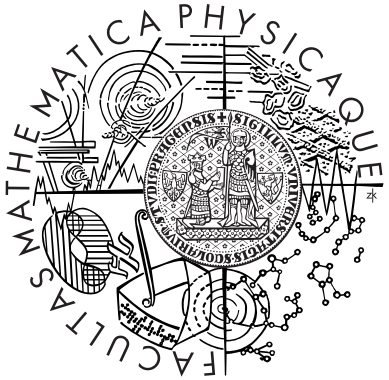
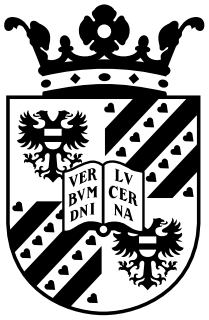
Rijksuniversiteit Groningen
Faculteit der Letteren

MASTER THESIS

Robust Parsing of Noisy Content

Joachim Daiber





Supervisor: Gertjan van Noord, Rijksuniversiteit Groningen
Daniel Zeman, Univerzita Karlova

Study programme: European Masters in Language
and Communication Technologies

Year: 2012–2013

Für meine Eltern Hans und Irmgard.

Acknowledgements

First and foremost, I am deeply indebted to Gertjan van Noord, whose invaluable suggestions and encouragement have kept me motivated to finish this work. I am equally thankful to my supervisor in Prague, Daniel Zeman.

The EM LCT masters program has been a great experience for me and it would not have been possible without the hard work and dedication by Markéta Lopatková and Vladislav Kuboň in Prague, Gosse Bouma in Groningen and Hans Uszkoreit, Valia Kordoni, Ivana Kruijff-Korbayova and Bobbye Pernice in Saarbrücken. I have learned invalueable lessons throughout these two years and I hope that many more students will be able to seize this opportunity.

Finally, I would like to thank Ke Tran Manh and Milos Stanojevic, who made my first year in Prague very enjoyable and who inspired and motivated me, and Sibel, who reminded me not to lose track of the bigger picture.

Joachim Daiber
Groningen, 2013

Contents

CHAPTER I

Introduction

Syntactic parsing is a central task in many natural language processing applications, and improvements in the complexity of popular parsing algorithms have been steadily increasing the task's utility. The most successful parsing algorithms today are based on statistical methods and require large manually annotated treebanks. Such treebanks are generally built from texts of one or a small number of textual domains, such as newswire texts in the case of the Wall Street Journal sections of the English Penn treebank. The performance of a parser on in-domain text, i.e. text from the same domain or from a domain similar to the domain of the training corpus, has improved steadily in recent years. However, out-of-domain text, i.e. text not from the training domain, and grammatically noisy text remain an obstacle and often lead to significant decreases in parsing accuracy. In this thesis, we focus on parsing of noisy content, as user-generated content in services like Twitter, so-called tweets. The term noisy content here refers to the orthographic quality of the texts. In corpora such as the Wall Street Journal sections of the Penn treebank, the textual content is collected from well-edited newspaper articles, which were produced under few restrictions and whose language is therefore particularly close to perceived or written orthographic standard. On the other hand, texts in services like Twitter, which are often called *user-generated content*, are produced under different conditions and restrictions. Usually, such texts are not proof-read and they are written quickly. Hence, there is an abundance of non-conventional orthography, punctuation and case. The additional constraints of this type of medium — tweets are restricted to 140 characters — lead to the frequent usage of abbreviations and other word formation processes that can help reduce the length of the text, such as the omission of vowels. Accordingly, we use the term *noisy* to denote such deviations from the conventions used in corpora of well-edited long-form articles.

As an example of what we regard as noisy text, consider the following sentence from the test section of our data set.

(1) ppl are thinking that i wil expect something from someone but #fact is i dont want anything because i wont expect anything from anyone

The example in (1) includes several of the issues mentioned previously: an abbreviation created by the omission of vowels (*ppl*), non-standard case, general deviations from conventional spellings (*wont*, *dont*, *wil*), as well as Twitter-specific tokens (*#fact*). Therefore, this thesis will be focused on methods for dealing with this kind of noise in the syntactic parsing task. We will compare various strategies for adaptation, and explore whether a text-normalization step based on machine translation techniques—as it has been successfully applied to other tasks such as machine translation and part-of-speech tagging (e.g. Kaufmann [2010])—can be used for syntactic parsing as well.

Furthermore, unsupervised methods, which do not require parallel data, will be explored, and we will investigate how such a preprocessing step can be integrated in a dependency parser model (MST parser, McDonald et al. [2005]). We will test our approach by comparing various parser configurations on existing data sets, as well as on a new data set for dependency parsing of noisy content. Both the existing data set and our new data set are based on tweets.

Common methods for domain adaptation in parsing work by retraining a parsing model with a focus on the target domain. While this is a reasonable method in most cases; in this thesis, we are particularly interested in the question of whether it is possible to achieve significant improvements for the parsing of noisy content without adapting the parser parsing model directly but by adapting the data before presenting it to the parser while leaving the parser model intact.

The structure of this thesis will be as follows: In Chapter 2, we introduce the background and context of this thesis. Chapter 3 discusses existing data sets for parsing of noisy content, followed by the introduction of a new data set for dependency parsing of noisy data. Unlike previous data sets, we also include textual normalization in the data set, and therefore, we also introduce a metric to be used in conjunction with the data set. Chapter 4 discusses and evaluates issues in part-of-speech tagging of noisy content. In Chapter 5, we discuss preprocessing steps and evaluate them on an existing data set, as well as on the data set introduced in Chapter 3. Finally, in Chapter 6, we present an overview of the results, perform an error analysis and discuss the results and their meaning.

CHAPTER II

Background and Related Work

In this chapter, we outline the necessary background in the relevant literature, lay out the relevance of the topic and embed it within the context of related topics. Firstly, we present an overview of dependency parsing and popular algorithms for this task. Secondly, we introduce the topic of domain adaptation in parsing. Thirdly, we briefly summarize the relationship between parsing and the noisy-channel model that will form the basis for the experiments we perform in the following chapters.

2.1 Dependency parsing

Dependency parsing is a method for the syntactic analysis of natural language text, based on the theory of dependency grammar. Rather than being a single unified theory, dependency grammar is a tradition encompassing several theories, which recently has gained additional popularity due to its value in various natural language processing tasks, such as relation extraction (e.g. Culotta and Sorensen [2004]) and machine translation (e.g. Shen et al. [2008]). Additionally, it is well-suited for the description of languages with free word order [Kübler et al., 2009].

Therefore, in the following section, we will briefly introduce dependency grammar, and give an overview of the differences among its various representations.

2.1.1 Dependency grammar

While dependency grammar has a rich history reaching back as far as the descriptions of the Sanskrit language, its modern tradition is believed to have begun with the French linguist Lucien Tesnière. In contrast to phrase structure grammars, in dependency grammars, the central means of describing linguistic structure is through words or tokens that are linked by directed dependency relations. The elements of a dependency relation are two words: the syntactically subordinate *dependent* and the word on which it depends, called the *head* (or *governor*). Furthermore, each dependency between two words is labeled by the *type* of their relation, such as *subject*, *direct object* or *attribute*. Consider, for example, the simple dependency tree in Figure 2.1.

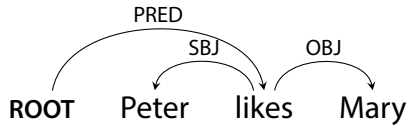


Figure 2.1: Simple dependency tree

In this example, *likes* is the head of both *Peter* and *Mary* and their dependency relations indicate that *Peter* is the subject (SBJ) and *Mary* is the object (OBJ) dependent on the verb *likes*, while the verb itself is a dependent of the artificial **ROOT** node. The **ROOT** node is inserted into the tree as a technical simplification to ensure that every word has a syntactic head.

Formal definition

As we will rely on these definitions in the following chapters and sections of this thesis, here we introduce the formal definition of dependency trees and their properties by following the formulation from [Kübler et al., 2009, p. 11].

A sentence S is a sequence of tokens $S = w_0 w_1 ... w_n$. The token w_0 is treated as the artificial root token, which we labeled **ROOT** in Figure 2.1. While the term token is commonly used as a synonym for a word in a sentence, depending on the language and conventions used, it does not have to be a full word but can represent single morphemes or punctuation.

Dependency relations are the relations between two words in the sentence. $R = \{r_1, ..., r_m\}$ is the finite set of dependency relation types that can hold between any two words. There are no further assumptions about the set R, and the final set of relations depends on the convention of the dependency format and the underlying linguistic theory. In the example in Figure 2.1, the dependency types are displayed as the labels of the arcs between the words: SBJ, OBJ and PRED.

In order to define dependency trees, which are a special type of dependency graphs, we first need to define dependency graphs. A dependency graph can be defined as the following (slightly adapting the definition from [Kübler et al., 2009, p. 12]): A dependency graph $G = (V, A)$ is a labeled directed graph consisting of nodes (vertices, V) and edges (arcs, A), such that the following holds for $S = w_0 w_1 ... w_n$ and the set of dependency types R:

1. $V \subseteq \{w_0, w_1, ..., w_n\}$
2. $A \subseteq V \times R \times V$
3. if $(w_i, r, w_j) \in A$ then $(w_i, r', w_j) \notin A$ for all $r' \neq r$

An edge $(w_i, r, w_j) \in A$ represents a dependency relation from the word w_i to the word w_j with the dependency type r. In this dependency relation, w_i is the head, and w_j is the dependent node. While the third condition disallows multiple dependency relations between two words, it does not restrict words from having more than one head. Most dependency grammar theories are based on dependency trees, which are a further restriction of dependency graphs that we will define now.

A *dependency tree* is defined as a *well-formed* dependency graph. A dependency graph G for a sentence S, and a dependency relation set R is called *well-formed* if it is a directed tree originating in the root node and if it has a spanning node set (i.e. a set of nodes containing all words of the sentence). A tree is an undirected graph in which any two nodes are connected by one and only one path. This means that a connected graph can only be a tree if it contains no cycles. In a directed tree, the edges between nodes are directed.

In a dependency graph, while it is possible for a word to have multiple heads, the tree property of dependency trees prohibits this. A node in the dependency tree can only have one single incoming edge. If a node in a dependency tree would have multiple heads, it would mean that there exist more than one paths between this node and the root node and other nodes common to both paths.

Projective and non-projective dependency trees

A further restriction on dependency trees is the distinction between projective and non-projective dependency trees. The following notations are required for the definition of projective and non-projective dependency trees:

- $w_i \rightarrow w_j$ indicates an unlabeled dependency relation between w_i and w_j in a tree $G = (V, A)$. Hence, $w_i \rightarrow w_j$ means that there is some dependency relation type $r \in R$ such that $(w_i, r, w_j) \in A$.
- $w_i \rightarrow^* w_j$ indicates a *reflexive transitive closure* of the dependency relation in a tree $G = (V, A)$. This is defined as $w_i \rightarrow^* w_j$ if and only if $i = j$ (*reflexive*) or both $w_i \rightarrow^* w_{i'}$ and $w_{i'} \rightarrow w_j$ hold for some $w_{i'} \in V$.

An edge in a dependency tree is projective if and only if $w_i \rightarrow^* w_k$ for all $i < k < j$ when $i < j$, or $j < k < i$ when $j < i$. Following this definition, an edge (w_i, r, w_j) is projective if there is a path from the head w_i to all the words between the two endpoints of the edge (i and j). Accordingly, a dependency tree $G = (V, A)$ is *projective* if all $(w_i, r, w_j) \in A$ are projective. A dependency tree that is not projective is *non-projective*.

As examples of projective and non-projective dependency trees, consider the two dependency trees in Figure 2.2 and Figure 2.3, which are dependency trees for commonly used English example sentence from the Penn treebank (both dependency trees are taken from [Kübler et al., 2009, p. 17]).

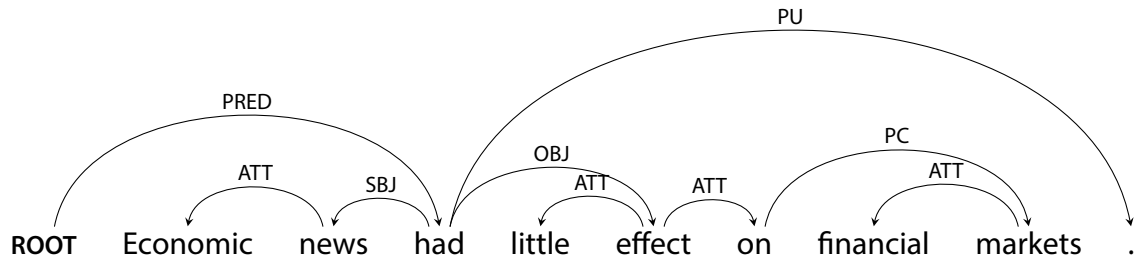


Figure 2.2: A projective dependency tree

In this representation of dependency trees, with all words at the same level, the dependency tree in Figure 2.2 can be drawn without crossing edges as all edges in this tree are projective. For the dependency tree in Figure 2.3, however, it is not possible to draw the tree without crossing edges. For the dependency relation between *hearing* and *on*, there is no path from the head of the relation *hearing* to the words in between the two end-points of the edge, namely the words *is* and *scheduled*. Hence, the edge between *hearing* and *on* is not projective.

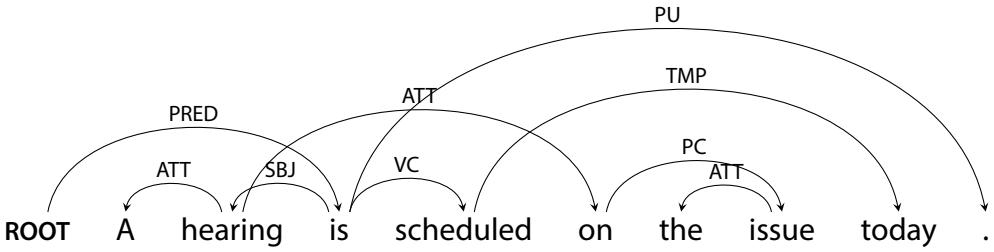


Figure 2.3: A non-projective dependency tree

2.1.2 Data-driven dependency parsing

The task of a dependency parsing algorithm is to find the dependency structure for the words in a sentence. In this dependency structure, words, including the artificial **ROOT** node, are connected by labeled dependencies. While the set of dependency labels is determined by the dependency representation format; most dependency

parsing algorithms are independent of the specific labels, and they can be trained for various types of dependencies.

Parsing in general, and specifically dependency parsing, are nontrivial tasks since the dependencies between the words of a sentence need to be determined while both adhering to linguistic and syntactic constraints, and at the same time ensuring that the resulting dependency graph is well-formed. This process is complicated by the tendency of natural language to be inherently ambiguous. For a given sentence, a parser may find multiple possible analyses, and as a result, it is required to make an optimal decision taking into account both local syntactic constraints as well as the complete analysis.

Data-driven methods for dependency parsing are based on machine learning techniques, most notably supervised methods. In supervised learning, a model is trained on a set of manually annotated instances. Using this trained model, the structure of new instances can be inferred. Hence, a data-driven parsing method must address two problems: First, in the *Learning* task, a parsing model M is learned from a set of annotated training instances (dependency trees). Then, in the *Parsing* or *Decoding* task, the parsing model M is used to produce the optimal dependency graph G for a sentence S.

2.1.3 Maximum-spanning tree parsing

The *learning* and *parsing* tasks are implemented in various ways by different parsers. A well-known algorithm is the Maximum Spanning Tree parser (MST, McDonald et al. [2005]). In the following section, we will briefly summarize how these two tasks are approached in the MST parser.

Learning: Online Large Margin Learning The *learning* task in this parser is approached as a supervised learning problem for structured output. The score of a dependency tree is factored as the sum of the scores of all edges in the tree. The score for each edge is calculated as the dot product of a feature vector for the edge, and the weight vector $\mathbf{w}$.

$$s(i,j) = \mathbf{w} \cdot \mathbf{f}(i,j)$$

Given a sentence $\boldsymbol{x}$, the score for a dependency tree $\boldsymbol{y}$ is:

$$s(\boldsymbol{x}, \boldsymbol{y}) = \sum_{(i,j) \in \boldsymbol{y}} s(i,j)$$

To determine the weight vector $\mathbf{w}$, a learning algorithm based on the Margin Infused Relaxed Algorithm (MIRA, Crammer and Singer [2003]) is used. MIRA is an online learning algorithm similar to the online version of the basic Perceptron learning algorithm [Rosenblatt, 1958]. Unlike a batch learning method, the online algorithm works by updating the weight vector $\mathbf{w}$ after each training instance it considers.

Pseudo-code for the MIRA algorithm is given in Algorithm 1. $dt(\boldsymbol{x})$ is the set of possible dependency trees for the sentence $\boldsymbol{x}$. The algorithm runs for N iterations, and during each of the iterations, it traverses all training instances, and determines the optimal weight with regard to the loss function L. In the case of dependency parsing, L is the number of words in the dependency tree with incorrect parent words compared to the correct tree. After N iterations, the intermediate weights stored in $\mathbf{v}$ are averaged, which has been shown to decrease the risk of the model overfitting to the training data [Collins, 2002].

Algorithm 1 The MIRA learning algorithm

Training data: $\tau = \{(x_t, y_t)\}_{t=1}^{T}$
$\mathbf{w}_0 \leftarrow 0$
$\mathbf{v} \leftarrow 0$
$i \leftarrow 0$
for $n : 1..N$ **do**
    for $t : 1..T$ **do**
        min $||\mathbf{w}^{(i+1)} - \mathbf{w}^{(i)}||$
        s.t. $s(x_t, y_t) - s(x_t, y') \geq L(y_t, y'), \forall y' \in dt(x_t)$
        $\mathbf{v} \leftarrow \mathbf{v} + \mathbf{w}^{(i+1)}$
        $i \leftarrow i + 1$
$\mathbf{w} \leftarrow \mathbf{v}/(N \cdot T)$

Decoding: Projective and non-projective trees In the MST parser, the task of finding the optimal parse tree is formulated as finding a maximum spanning tree for the words of the input sentence, by using the edge scores, which were introduced above as weights within the graph. For the projective and the non-projective case, separate decoding algorithms are used. The projective decoding algorithm is the Eisner algorithm [Eisner, 1996], which is a frequently used $O(n^3)$ algorithm for maximum projective spanning trees. In the non-projective case, the search for a suitable parse tree is performed by searching for a maximum spanning tree in a directed acyclic graph using the Chu-Liu-Edmonds algorithm [Chu and Liu, 1965, Edmonds, 1967]. Specifically, an implementation of the algorithm for dense graphs which has complexity $O(n^2)$ is used.

2.2 Parsing and domain adaptation

2.2.1 Domain adaptation

According to the Concise Oxford Dictionary of Linguistics [Matthews, 2013], a domain in the cultural or in another setting is a situation, or a text form in which different forms of speech may be appropriate; such as the different forms of speech in the domain of a law court or of sports commentary, compared to the domain of a family at home. In natural language processing, domains are conventionally delimited collections of texts or literary genres such as news wire articles or biomedical literature. However, especially since the emergence of the World Wide Web has weakened traditional forms of publishing and categorization, the definition and delimitation of domains has become more difficult. Some authors go as far as arguing that on the World Wide Web, each document is its own domain [e.g. McClosky et al., 2010]. Methods for domain adaption vary in the extent to which strict boundaries between domains are assumed. In the following section, we will introduce the most widely used methods for domain adaption in parsing.

2.2.2 Methods for domain adaption

In the relevant literature, it is commonly acknowledged that parsing accuracy degrades when a parsing model trained on a certain domain is applied to a different domain. In order to investigate this issue, various methods and specific task settings have been brought forward. In this section, we will briefly summarize the most important findings and methods; and will also outline how they relate to the topic of this thesis.

One of the first issues requiring to be addressed in domain adaptation is the lack of a precise definition of what a domain constitutes. Approaches differ in whether they assume that domains are given or in whether they attempt to select data similar to a target domain automatically.

Single source parser adaptation

The CoNLL 2007 Shared Task on dependency parsing [Nivre et al., 2007a] contained a track on domain adaptation, in which the participants were asked to produce the best possible parsing results across domains. Participants were provided with a large syntactically annotated corpus from the source domain as well as data from three target domains (biomedical abstracts, chemical abstracts and parent-child dialogues).

Like the source domain data, the target domain data contained full dependency trees; however, the set of biomedical abstracts was only used as a development set, while the set of chemical abstracts was used as a test set. Additionally, large unlabeled data sets, i.e. data sets without any annotation, were provided as training data for each of the target domains. Hence, in this setting only a single source domain, in this case the Wall Street Journal sections of the Penn Treebank, was used.

The most successful systems participating in the domain adaptation part of the CoNLL 2007 Shared Task used two major approaches: Firstly, in feature-based approaches, the features of the parsing system were either reduced to features mostly valid across various domains, or features from the source domain were transferred to the target domain [Nivre et al., 2007a]. And secondly, in ensemble-based approaches, several parsers are trained, and run; and then their output was combined in a new classifier. Other approaches included tree revision rules and filtering of the training set based on similarity to the target domain.

Other methods that have gained in popularity recently are *self-training* and *uptraining*. *Self-training* is a method for effective parser adaptation that was introduced by McClosky et al. [2006]. In self training, a parser is learned on labeled data, and then it is used to annotate unlabeled data, possibly from the target domain. Together with the original labeled data, the parsed unlabeled data is used afterwards as training data for a new model. This process can be repeated several times. Intuitively, since errors can propagate into the parsing model, one may expect this method to degrade parsing accuracy. However, the authors found that the self-training method provided a 12% error reduction over the previous best result of Wall Street Journal parsing. The authors' error analysis suggests that improvements were not obtained as a result of better unknown word handling; and furthermore, the accuracy for sentences of a length between 20 and 50 words improved in general.

Uptraining [Petrov et al., 2010] is a method for domain adaptation similar to *self-training*. This method is focused more on deterministic parsers, such as shift-reduce dependency parsers. In *uptraining*, a deterministic parser is trained on the output of a slower but more accurate constituent parser. The authors demonstrate that a deterministic parser trained on 100.000 questions parsed by a more accurate parser provides results comparable to a deterministic parser trained on 2.000 manually annotated questions.

For closely related languages, Zeman and Resnik [2008] describe an approach to parser adaptation to a new language in the special case that the target language—which has only few resources available—is related to the source language. The authors

demonstrate that a de-lexicalization method works well in an experiment with data from Danish and Swedish. In de-lexicalization task, the words of the source language are replaced by their morphological tags in the training data, and the same tagset is used when applied to the target language. Parsing the Swedish data with a de-lexicalized Danish parsing model achieved parsing performance equivalent to a parser trained on 1546 Swedish gold trees.

Multiple source parser adaptation

McClosky et al. [2010] take a different approach for the parser adaptation problem by introducing a new task, which include baseline systems for what they call *multiple source parser adaptation*. In this setup, a system is trained on corpora from various domains, and it learns the plain parsing models, as well as models of domain differences and their influence on parsing accuracy. Given this knowledge, the parser applies a linear combination of plain parsing models to new input.

Automatic selection of relevant training data

The approaches to domain adaption mentioned so far involve the assumption that domains are given in advance. Plank and van Noord [2011] evaluate methods to select training data similar to the target domain automatically, by using an unsupervised method based on topic modeling. This method does not rely on predefined domains. Instead, in order to gather relevant data for the particular target domain from the source domain corpus, measures of domain similarity are used.

2.3 Parsing and the noisy-channel model

2.3.1 The noisy-channel model

The noisy-channel model [Shannon, 1948] is a mathematical model for communication that has been successfully applied to a wide variety of natural language processing tasks. The original model was motivated by the wish to transmit the maximum possible amount of information over a noisy means of communication [Manning and Schütze, 1999, p. 60]. Today, this model is the basis for a number of successful approaches to problems such as machine translation and automatic spelling correction. In the noisy-channel model, a message is generated by the information source and passed trough a noisy channel. The receiver is given the resulting noisy data; and the receiver's goal is to recover, or in other words to *decode*, this noisy data in order to determine the original message. Given observed data O with the original source form

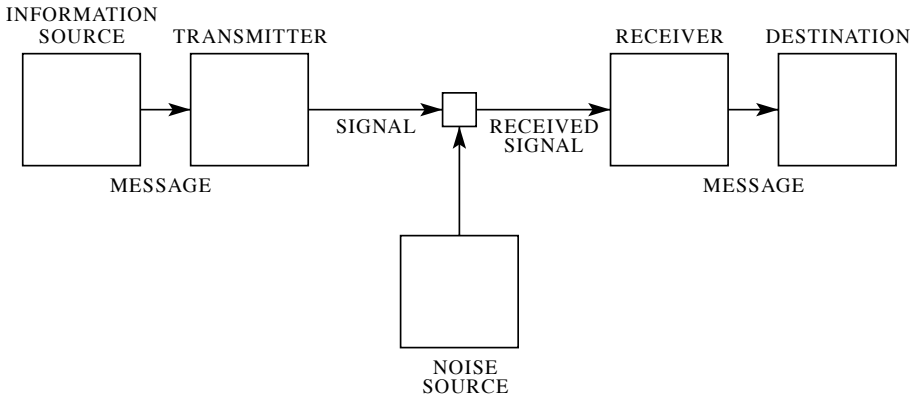


Figure 2.4: Schematic diagram of a general communication system [Shannon, 1948, p. 2]

S for which there exists some prior knowledge, this process is typically modeled by using Bayes' rule [Lease et al., 2006]:

$$\hat{s} = \arg\max_{S} \mathrm{P}(S|O) = \arg\max_{S} \mathrm{P}(O|S)\,\mathrm{P}(S) \tag{2.1}$$

2.3.2 Sentence comprehension under noisy input

In the psycholinguistics literature, a problem that is closely related to parsing is the problem of human sentence comprehension. While most research in this area assumes perfectly-formed input; Levy [2008] proposes a noisy-channel based model of human sentence comprehension that can account for some of the outstanding problems in sentence processing. The model uses a generative probabilistic grammar. In the standard case that the input string $\mathbf{w}$ is fully known, the grammar G can be used to find the best parse T for $\mathbf{w}$ using

$$\arg\max_{T} \mathrm{P}_G(T|\mathbf{w}) \tag{2.2}$$

However, in the case that the input string is not known, the formulation changes to the following: Given some noisy evidence I and using the Bayes' rule to find the posterior for $\mathrm{P}_G(T|\mathbf{w})$ and $\mathrm{P}_G(T|I)$:

$$\mathrm{P}_G(T|I) = \frac{\mathrm{P}(T,I)}{\mathrm{P}(I)} \tag{2.3}$$

$$\propto \sum_{\mathbf{w}} \mathrm{P}(I|T,\mathbf{w})\,\mathrm{P}(\mathbf{w}|T)\,\mathrm{P}(T) \tag{2.4}$$

While not directly applying this model to parsing, Levy [2008] performs a psycholinguistics experiment in which he assumes a noisy environment distorting an original utterance. A comprehender is presented a sentence $\mathbf{w}^*$, which is known by the researchers. However, the specific noisy representation I of this sentence is not known. Based on the formula in (2.4), for the probability of the comprehender's understood sentence $\mathbf{w}$, we have:

$$\mathrm{P}_G(\mathbf{w}|I) \propto \sum_{\mathbf{w}} \mathrm{P}(I|T, \mathbf{w})\, \mathrm{P}(\mathbf{w}|T)\, \mathrm{P}(T) \tag{2.5}$$

In the controlled experiment of Levy [2008], the relevant probability distribution is $\mathrm{P}(\mathbf{w}|\mathbf{w}^*)$, which is given as

$$\mathrm{P}(\mathbf{w}|\mathbf{w}^*) = \int_I \mathrm{P}_C(\mathbf{w}|I, \mathbf{w}^*)\mathrm{P}_T(I|\mathbf{w}^*)\, \mathrm{d}I \tag{2.6}$$

Then, since the comprehender does not know $\mathbf{w}^*$, it is assumed that $\mathbf{w}^*$ and $\mathbf{w}$ are conditionally independent. Applying Bayes' rule to 2.6, the following formulation is proposed (P_C is the probability distribution of the comprehender):

$$\mathrm{P}(\mathbf{w}|\mathbf{w}^*) = \int_I \frac{\mathrm{P}_C(I|\mathbf{w})\mathrm{P}_C(\mathbf{w})}{\mathrm{P}_C(I)} \mathrm{P}_T(I|\mathbf{w}^*)\, \mathrm{d}I \tag{2.7}$$

$$= \mathrm{P}_C(\mathbf{w}) \int_I \frac{\mathrm{P}_C(I|\mathbf{w})\mathrm{P}_T(I|\mathbf{w}^*)}{\mathrm{P}_C(I)}\, \mathrm{d}I \tag{2.8}$$

$$\propto \mathrm{P}_C(\mathbf{w})Q(\mathbf{w}, \mathbf{w}^*) \tag{2.9}$$

$Q(\mathbf{w}, \mathbf{w}^*)$ in (2.9) is proportional to the integral in (2.8) and given some of the experiment's assumptions; it is a symmetric, non-negative function of $\mathbf{w}$ and $\mathbf{w}^*$. In the rest of the paper, $Q(\mathbf{w}, \mathbf{w}^*)$ is implemented as a simple Levensthein distance measure calculated via a token-based finite-state automaton.

One of the arguments that is presented as supporting this model comes from a simple prediction the model makes: the prior knowledge $\mathrm{P}_C(\mathbf{w})$ of the comprehender can overwrite the actual linguistic input. Hence, a sentence can be interpreted as meaning something else than it was originally expressed to mean. This is a necessary requirement of such a model: A human copy editor, for example, is without a question, able to read and correct erroneous writing. Non-noisy models of sentence processing are able to cover this issue in cases where the sentence is ungrammati-

cal; however, the noisy model can also account for cases where the sentence is fully grammatical but the comprehender can still not process the sentence. This is the case in certain garden path sentences, as in the following example from Levy [2008]:

(1) a. While the man hunted the deer ran into the woods.
    b. While the man hunted it the deer ran into the woods.

In this case, both sentences are grammatical sentences. For the sentence in (1), there is a grammatical reading; however, it is ambiguous whether *the deer* is the object of the verb *hunted*, or the subject of the verb *ran*. In the case that it is read as the object of *hunted*, the *ran* will miss a subject, and the comprehender would re-read the sentence as it "leads nowhere". However, experiments in which participants are given this sentence have shown that a large number of participants initially perceive *the deer* as the object of *hunted*. Non-noisy models of sentence processing would not predict this; however, it can be explained by the noisy model that can assume an underlying sentence such as the sentence in (1).

Note the similarity of the final model in (2.9) to well-established noisy-channel methods for spelling correction, such as Kernighan et al. [1990], where a correction c for a text t is found by maximizing $\mathrm{P}(c)\,\mathrm{P}(t|c)$.

2.4 Parsing and social media content

A prime example of noisy content is the content of social media services such as Twitter. There has been a surge of interest in Twitter since it provides a steady, real-time source of personal and news-like content from users all over the globe. Twitter data has been used in a multitude of tasks, such as predicting stock market trends or social and political science research. It has also been used for predicting the spread of infectious diseases based on users' utterances and exchanges indicating symptoms of illness (e.g. Hirose and Wang [2012] and Sadilek et al. [2012]).

Foster et al. [2011] treat parsing of Twitter content as a domain adaptation task and use a combination of self-training and uptraining to adapt a WSJ-trained version of the Malt parser [Nivre et al., 2007b] for use on Twitter data. Their adaptations are tested on a small treebank. They found that while the unadapted parser had low performance, adaptation via uptraining and self-training improved parsing performance significantly. Additionally, it was observed that performance of the necessary subtasks such as part-of-speech tagging already degraded significantly on the Twitter data.

2.5 Text-normalization of social media content

In other natural language processing tasks, such as machine translation or entity linking, researchers have approached the processing of social media content by performing unsupervised or semi-supervised text-normalization as a preprocessing step before an actual application. There are both supervised and unsupervised methods for performing this task. In the following section, we will give a brief overview of these methods and touch upon their restrictions.

The majority of unsupervised models for text-normalization are based on the noisy-channel model. For the related task of text message normalization, Cook and Stevenson [2009] present a method in which they model $P(S|O)$, i.e. the probability of the original source form S given the observed data O, by analyzing and modeling common word formation processes in their data. These processes include stylistic variations, subsequence abbreviations and prefix clippings. Han and Baldwin [2011] propose a simple normalization method, in which a classifier detects *real* out-of-vocabulary tokens and a mixture of features similar to Cook and Stevenson [2009] is used to find the best suggested correction for a token classified as a misspelling of an in-vocabulary word. The authors simplified and improved this system further in Han et al. [2012], replacing it by a dictionary substitution method. The substitution dictionary can be gathered before the actual system is applied. We will return to this method in Section 5.1.

Starting with Aw et al. [2006] for text message normalization and continuing with Kaufmann [2010] for Twitter messages, the text-normalization task has also been treated as a supervised translation task from *noisy* language into *clean* language. Reducing this problem to the problem of machine translation enables the use of a set of existing methods and tools for statistical machine translation, such as the Moses toolkit [Koehn et al., 2007] for phrase-based machine translation. Based on the resources available from Aw et al. [2006] and an additional, manually created parallel corpus, Raghunathan and Krawczyk [2009] perform text message normalization by using the Moses toolkit.

Kaufmann [2010] performs experiments showing that with minimal additional preprocessing, the same parallel data that is used for text message normalization can also be used to normalize Twitter data. This data consists of parallel corpora of manually normalized text messages. As with the other papers, the method is evaluated using the BLEU evaluation metric [Papineni et al., 2002].

CHAPTER III

Data Sets for Parsing of Noisy Content

In this chapter, we briefly introduce existing data sets relevant to the topic of this thesis and introduce a new data set for dependency parsing of noisy content. With it, we introduce an evaluation metric and illustrate its relationship to common evaluation metrics for dependency parsing.

3.1 Data sets for parsing of noisy content

In order to evaluate domain adaption to Twitter data, Foster et al. [2011] created a small test treebank consisting of syntactically annotated sentences taken from tweets. The treebank was created as a selection of 519 sentences from a corpus of 60 million tweets from Bermingham and Smeaton [2010], which was based on work on detecting themes and topics in Twitter messages. This corpus is restricted to tweets classified as belonging to any of 50 topics, such as politics, business, sports and entertainment. Since the focus of this thesis is on noisy content, we aim not to restrict ourselves to fixed topics and create a small treebank which aims to be representative of everyday Twitter language.

3.2 Dependency parsing test set

To achieve this goal, we have created a new dependency parsing test set consisting of sentences retrieved from Twitter messages that were manually annotated with dependency structure. Here, we provide an overview of the annotation decisions and elaborate on the means of collecting the data with the goal of creating a representative sample of the language used in Twitter messages.

3.2.1 Acquisition and corpus statistics

We collected all tweets within a period of 24 hours from January 07, 2012 00:00 until 23:59 GMT. To avoid possible biases of language identification and topic classification tools towards well-formed language, we manually classified the tweets in random

order into English and non-English tweets until we reached a reasonably sized corpus of tweets classified as English. We then manually split this corpus into sentences and randomly selected 250 sentences as a development set and 250 sentences as a test set. Properties of the test and development set compared to the test and development set of Foster et al. [2011] are listed in Table 3.1. The out-of-vocabulary rate for tokens and types has been calculated against the English dictionary of the GNU Aspell spell checker.[1]

		Sentence length			**Token length**			**OOV rate**	
Corpus	# sent.	Mean	Med.	Std	Mean	Med.	Std	Tokens	Types
Dev.	250	9.608	8	5.65	4.27	4	2.61	0.332	0.440
Test	250	9.420	8	5.30	4.40	4	2.65	0.340	0.444
Foster dev.	269	11.14	10	6.41	4.00	4	2.405	0.344	0.328
Foster test	250	11.36	10	6.79	4.10	4	2.462	0.342	0.331

Table 3.1: Properties of our test and development sets compared with the data set from Foster et al. [2011]

3.2.2 Annotation decisions

The sentences in both the development and test set were normalized to standard spelling; missing words that are necessary for the comprehension of the sentence were inserted; and abbreviations and contractions were extended when it was deemed relevant. The goal of this normalization was to leave the original tokens intact and not to replace them by their normalizations directly. Hence, we keep both the original tokens and the normalized versions of the sentences and create word alignments showing the relation between the original and normalized text.

Text-normalization

Abbreviations and slang expressions have been extended whenever necessary for syntactic reasons. Examples include instances such as “cu”, used as the short form of *see* and *you*, which as a single token would include both the verb and the object of the sentence.

Punctuation has only been inserted if it was necessary to disambiguate the meaning of the sentence, and capitalization has only been corrected to preserve the case of proper nouns. Emoticons, such as ‘:-)‘, have been kept intact and tagged with the part-of-speech tag `UH` (interjection). During the annotation, a frequently occurring

[1] `http://aspell.net/`

phenomenon was the occurrence of zero copula, i.e. a copula verb is not realized in the sentence.

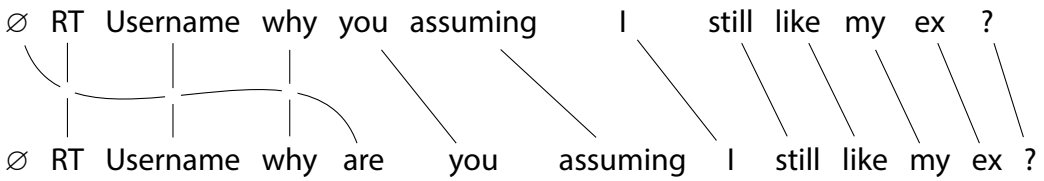


Figure 3.1: Example of a zero copula annotation

We annotated these occurrences by inserting the copula verb in the normalized version of the sentence as illustrated in an example sentence and its aligned normalization in Figure 3.1.

The most frequent insertions and replacements in the test section of the data set are listed in Table 3.2.

Replacement	**Frequency**	**Insertion**	**Frequency**
u -> you	12	∅ -> are	6
∅ -> are	6	∅ -> is	2
its -> it 's	4	∅ -> this	1
& -> and	4	∅ -> a	1
ur -> your	3		
ta -> to	3		
ppl -> people	3		

Table 3.2: Common replacements and insertions in the test section of the data set

Dependency annotation

The normalized tokens of the test and development set were automatically parsed using a generative phrase structure parser,[2] and then converted to dependencies. Both part-of-speech tags and dependency annotations were then manually corrected in two passes. The dependency annotations use the guidelines and format from the CoNLL-X shared task on multilingual dependency parsing [Buchholz and Marsi, 2006]. Figure 3.2 depicts part of a gold standard dependency graph including the alignments to the original tokens.

[2] `https://code.google.com/p/berkeleyparser/`

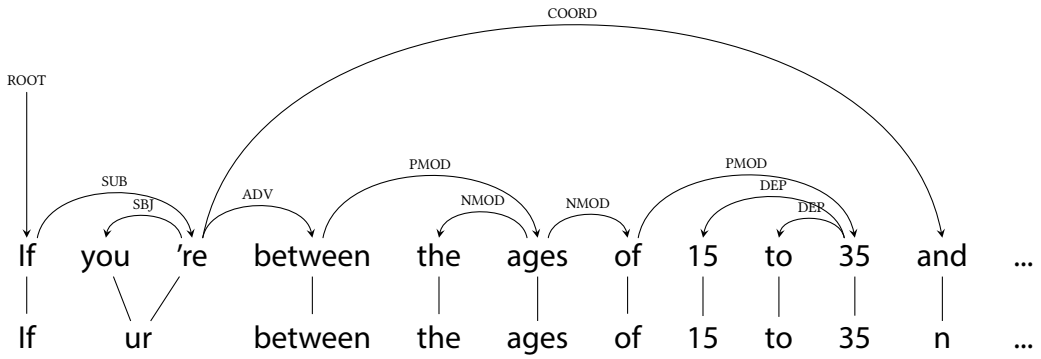


Figure 3.2: Example of a gold standard dependency graph with alignments

Twitter-specific syntax

For Twitter-specific syntax in the corpus, the following guidelines were applied: User names at the beginning of a sentence that do not fulfill a syntactic role, e.g. as the subject, are attached to the main verb using the `DEP` dependency relation, which in the CoNLL format is the dependency type for unclassified relations.

(1) Username happy birthday Sir

(2) Username is my guest today

The occurrence of *Username* in (1) is an example of this phenomenon. On Twitter, the usage of a user name at the beginning of a tweet serves the purpose of notifying the person that the utterance is directed at them. In (2) in contrast, the user name is part of the actual sentence.

RT and similar markers (*RT* indicates a *retweet*, akin to a direct quotation) are equally attached to the main verb using the dependency relation `DEP`.

(3) RT Username Why can't my summer vacation be like Phineas & Ferbs?

Hashtags such as `#decisionsdecisions` in (4) indicating the topic of a tweet are equally attached to the main verb using the `DEP` dependency type.

(4) thinking about going back to school tomorrow #decisionsdecisions

3.3 Evaluation metrics

Since alignments between the gold standard and the original tokens are included and both insertions and deletions should be possible, we cannot directly use accuracy as a

measure for evaluation. To calculate the unlabeled or labeled accuracy for two dependency trees, the head and dependency relation of each word in the gold dependency tree are compared to the head and dependency relation of the corresponding word in the predicted dependency tree. In the case of data set introduced here, there may not be a direct one-to-one correspondance between the predicted tree and the gold tree. Hence, we allow the parser to make any insertions, deletions and modifications to the tokens under the assumption that it provides an alignment between the modified tokens and the original tokens. The evaluation is then performed using a metric based on precision and recall values calculated using these alignments. This metric will be introduced and defined in the remainder of this section.

3.3.1 Aligned precision and recall

Based on the normalized side of the gold standard and the parser's aligned prediction, we then calculate precision, recall and F_1 score for dependencies (3.1–3.4). We base the evaluation metric on the standard definitions of precision and recall, which is widely used in information retrieval and various other fields. In (3.1) and (3.2), *TP* is the number of true positive results, *FP* is the number of false positive results, and *FN* is the number of false negative results. The F_1 measure is the harmonic mean of precision and recall.

$$\mathit{precision} = \frac{\mathit{TP}}{\mathit{TP} + \mathit{FP}} \tag{3.1}$$

$$\mathit{recall} = \frac{\mathit{TP}}{\mathit{TP} + \mathit{FN}} \tag{3.2}$$

$$F_\beta = (1 + \beta^2) \cdot \frac{\mathit{precision} \cdot \mathit{recall}}{\beta^2 \cdot \mathit{precision} + \mathit{recall}} \tag{3.3}$$

$$F_1 = 2 \cdot \frac{\mathit{precision} \cdot \mathit{recall}}{\mathit{precision} + \mathit{recall}} \tag{3.4}$$

In the following, we formalize the metric based on these definitions of precision and recall. As introduced previously in Section 2.1.1, given a sentence $S = w_0 w_1 ... w_n$, where w_0 is the root node, a dependency graph is a set $G = \langle V, A \rangle$ of vertices V and arcs A. A dependency tree is a dependency graph exhibiting the property of being *well-formed*.

Definition

Let P be the set of quintuples $\langle S^O, D_P, D_G, a_P, a_G \rangle$ for the whole test set. Each quintuple consists of the following elements:

– the original sentence S^O

– a predicted dependency tree $D_P = \langle V_P, A_P \rangle$

– a gold dependency tree $D_G = \langle V_G, A_G \rangle$

– an alignment function a_P for the predicted tokens

– an alignment function a_G for the gold tokens

A sentence has the form of a sequence of tokens $S = w_0, w_1, ..., w_n$. For each parsed dependency tree S^O is the sequence of original, non-normalized tokens. And, given a predicted dependency tree D_P, and a gold dependency tree D_G; S^P is the sequence of tokens for the dependency tree D_P, and let S^G is the sequence of tokens for the gold dependency tree D_G.

There are two alignment functions a_G and a_P, which map the gold tokens in S^G to the original tokens in S^O, and the predicted tokens in S^P to the original tokens in S^O. The alignment function $a_G : j \rightarrow i$ maps a token S^G_j from the gold part to its corresponding original token S^O_i, and the alignment function $a_P : j \rightarrow i$ maps a token S^P_j from the predicted part to its corresponding original token S^O_i.

In the case of an insertion, the new token cannot be aligned to any of the original tokens in S^O. Therefore, insertions are modeled as a mapping to an artificial `NULL` token. For example, assume that S^O is the sentence "u da boss", and the predicted sentence S^P is the sentence "you are the boss". In this case, a_P would provide the following mapping (we replace indices with their corresponding tokens for readability and use $\rightarrow$ to indicate an alignment): *you* $\rightarrow$ *u*, *are* $\rightarrow$ `NULL`, *the* $\rightarrow$ *da* and *boss* $\rightarrow$ *boss*.

Based on the set P of quintuples $\langle S^O, D_P, D_G, a_P, a_G \rangle$, we calculate the total number of true positive (*TP*), false positive (*FP*) and false negative (*FN*) dependency relations as follows:

For each gold dependency tree $D_G = \langle V_G, A_G \rangle$ and each predicted dependency tree $D_P = \langle V_P, A_P \rangle$, let M_G and M_P be the set of dependency relations mapped to the original tokens in S^O:

$$M_G = \{\langle a_G(i), a_G(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_G\} \tag{3.5}$$

$$M_P = \{\langle a_P(i), a_P(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_P\} \tag{3.6}$$

Then, we can determine the set of *true positive* dependencies as the intersection of M_G and M_P:

$$TP = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |M_G \cap M_P| \tag{3.7}$$

The set of *false positive* dependencies is the set of all predicted dependencies *without* all correct dependencies:

$$FP = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |M_P \setminus M_G| \tag{3.8}$$

And, the set of *false negative* dependencies is the set of all gold dependencies *without* the predicted dependencies:

$$FN = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |M_G \setminus M_P| \tag{3.9}$$

Labeled dependencies

While the presented measure is for unlabeled dependency relations, it can straightforwardly be extended to labeled dependencies by replacing the head-modifier pair in M_P and M_G with a triple also containing the dependency type:

$$M'_G = \{\langle a_G(i), r, a_G(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_G\} \tag{3.10}$$

$$M'_P = \{\langle a_P(i), r, a_P(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_P\} \tag{3.11}$$

Issues

Note that the definition of M_G and M_P in (3.5) and (3.6) involve a simplification: a dependency relation a is treated as equivalent to another dependency relation b if the head of a maps to the same original token as the head of b and if the dependent of a maps to the same original token as the dependent of b. This assumption makes the computation of the score straightforward and does not require the predicted tokens and gold tokens mapping to the same original token to be aligned individually, which would require the use of heuristics.

One problem, however, is that this simplification may produce incorrect results in a very specific case: As an example, assume that $S^O = \langle$ ur, da, boss $\rangle$ and the predicted tokens are equivalent to the gold tokens $S^P = S^G = \langle$ you, are, the, boss $\rangle$. In this case, both a_P and a_G would consist of the following mapping: *you* $\rightarrow$ *ur*, *are* $\rightarrow$

ur, *the* → *da* and *boss* → *boss*. Accordingly, M_G and M_P would contain the following pairs (replacing indices with tokens for readability, and in the format $\langle$head, modifier$\rangle$ as before):

$$M_P = M_G = \{\langle \mathsf{ROOT}, \mathsf{ur}\rangle, \langle \mathsf{ur}, \mathsf{ur}\rangle, \langle \mathsf{ur}, \mathsf{boss}\rangle, \langle \mathsf{boss}, \mathsf{da}\rangle\} \tag{3.12}$$

The problem with using this set for comparison is that it is not able to distinguish which of the tokens mapping to ur is part of the dependency relation. For example, the set M_P can be produced equally using the following incorrect dependency tree over the normalized tokens $\langle$ you, are, the, boss $\rangle$:

$$\{\langle \mathsf{ROOT}, \mathsf{you}\rangle, \langle \mathsf{you}, \mathsf{are}\rangle, \langle \mathsf{you}, \mathsf{boss}\rangle, \langle \mathsf{boss}, \mathsf{the}\rangle\} \tag{3.13}$$

While this means that in very specific cases, this measure can count an incorrect dependency tree as correct, we do not consider this case very likely and prefer this simplification to the problems that would be introduced by heuristically aligning tokens mapping to the same original token. This problem can be mitigated further by evaluating labeled dependencies since in this case, additional to the dependency edge, the type of the dependency relation is also considered.

Relationship to unlabeled and labeled accuracy

In this thesis, aligned precision and recall is applied in cases where the parser is allowed to insert, delete or split any tokens where necessary. However, there is also the case that neither the gold standard nor the parser use any insertions or deletions. In this case, the mapping between the tokens and both the gold dependency tree and the predicted dependency tree is a 1-to-1 mapping. This case occurs for example, when evaluating a standard CoNLL format data set like the test set from Foster et al. [2011] against a parser that does not insert, split or delete tokens. In this specific case where only 1-to-1 mappings exist between the original tokens and nodes of the gold and predicted tree, the unlabeled and labeled aligned F_1 scores are equivalent to unlabeled and labeled accuracy. We will show this in the following section for the case of labeled dependencies. The case of unlabeled dependencies is analogous.

As introduced previously, A_G and A_P are the sets of dependency edges $\langle w_i, r, w_j\rangle$ for the dependency trees D_G and D_P, where each edge represents a dependency relation with label r between the word w_i and the word w_j.

For the set of gold and predicted dependency trees, the labeled accuracy can be

calculated as follows:

$$\textit{accuracy} = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} \frac{|A_G \cap A_P|}{d} \tag{3.14}$$

The calculation of accuracy assumes that there is a 1-to-1 mapping between the input tokens and the nodes of the gold dependency tree and the predicted dependency tree, hence $d = |A_G| = |A_P|$. Assuming w_i and w_j are unique identifiers, the formula can be re-written as:

$$\textit{accuracy} = \frac{|C_G \cap C_P|}{|C_G|} = \frac{|C_G \cap C_P|}{|C_P|} \tag{3.15}$$

where C_G and C_P are the sets of dependency relations for the whole corpus:

$$C_G = \bigcup_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} A_G \tag{3.16}$$

$$C_P = \bigcup_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} A_P \tag{3.17}$$

Hence, the accuracy is the percentage of dependency relations that were predicted correctly.

In our definition of aligned precision and recall, we assume two alignment functions a_G and a_P and the two sets M'_P and M'_G for labeled dependencies were defined as:

$$M'_G = \{\langle a_G(i), r, a_G(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_G\} \tag{3.18}$$

$$M'_P = \{\langle a_P(i), r, a_P(j) \rangle \mid \langle w_i, r, w_j \rangle \in A_P\} \tag{3.19}$$

Since we are considering only the case, where there is a 1-to-1 alignment between the original tokens and the gold and predicted nodes, a_P and a_G are 1-to-1 functions (injective functions). Because a_P and a_G are injective functions, M'_G and M'_P can equally be formulated as $M'_G = A_G$ and $M'_P = A_P$ in this particular case.

Technically, A_G and A_P would not necessarily have to contain the exact same tokens (w_i). For example, a predicted token could be normalized differently than in the gold standard dependency tree. However, since a_G and a_P are injective functions, it is possible to simply replace each token in A_G and A_P with its corresponding original token according to a_P and a_G to ensure equivalence.

Accordingly, the definitions of *TP*, *FP* and *FN* for the case that there are only 1-to-1

alignments are equivalent to the following:

$$TP = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |A_G \cap A_P| = |C_G \cap C_P| \tag{3.20}$$

$$FP = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |A_P \setminus A_G| = |C_P \setminus C_G| \tag{3.21}$$

$$FN = \sum_{\langle S^O, D_P, D_G, a_P, a_G \rangle \in P} |A_G \setminus A_P| = |C_G \setminus C_P| \tag{3.22}$$

Using the definitions for *TP*, *FP* and *FN* from above, *precision* and *recall* are:

$$precision = \frac{TP}{TP + FP} = \frac{|C_G \cap C_P|}{|C_G \cap C_P| + |C_P \setminus C_G|} = \frac{|C_G \cap C_P|}{|C_P|} \tag{3.23}$$

$$recall = \frac{TP}{TP + FN} = \frac{|C_G \cap C_P|}{|C_G \cap C_P| + |C_G \setminus C_P|} = \frac{|C_G \cap C_P|}{|C_G|} \tag{3.24}$$

Recall that the definition of accuracy was $accuracy = \frac{|C_G \cap C_P|}{|C_G|} = \frac{|C_G \cap C_P|}{|C_P|}$, hence we have $precision = recall = accuracy$. Since the F_1 score is the harmonic mean of both values, in the particular case that there are only 1-to-1 alignments, we have further:

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.25}$$

$$= 2 \cdot \frac{accuracy \cdot accuracy}{accuracy + accuracy} \tag{3.26}$$

$$= \frac{2 \cdot accuracy \cdot accuracy}{2 \cdot accuracy} \tag{3.27}$$

$$= accuracy \tag{3.28}$$

3.3.2 Statistical significance

To substantiate the effect of any observed metric gains on the data set, statistical significance tests must be performed. Statistical significance testing is used to exclude the possibility that a system's victory over another system on a given data set occurred merely by chance and does not constitute a real improvement in performance. For commonly used metrics, there is conventionally a rule-of-thumb stating how much of a metric gain will constitute a significant result (e.g. 0.4 F_1 score improvement in parsing and 0.5 BLEU score improvement in machine translation, Berg-Kirkpatrick et al. [2012]). However, while these rules-of-thumb have been demonstrated to be reasonable under strict constraints [Berg-Kirkpatrick et al., 2012], it is generally better to perform full significance testing against a baseline system to ac-

count for other complex factors such as the composition and size of the data set. Additionally, since we use a non-standard metric, such rules-of-thumb would not apply to the experiments we perform.

Hypothesis testing using the bootstrap procedure

Significance testing is performed using the bootstrap method [Efron and Tibshirani, 1993], which provides the advantage of being applicable to any given metric. The goal of hypothesis testing is to answer the question if, given the output of system A and system B on a small test set $x = x_1, ..., x_n$ where system A beats system B by $\delta(x)$, the result will hold on a large population of data. Hence, the goal is to estimate the likelihood $\mathrm{P}(\delta(X) > \delta(x)|H_0)$, in which X is a random variable over possible test sets, each of the same size as the data set, and H_0 is the null hypothesis that system A is not better than system B on the whole population. By convention, with a value of $\mathrm{P}(\delta(X) > \delta(x)|H_0) < 0.05$, the metric improvement $\delta(x)$ is considered significant. A low value of $\mathrm{P}(\delta(X) > \delta(x)|H_0)$ indicates that it is unlikely for the observed metric improvement to occur while system A is not actually better than system B. $\mathrm{P}(\delta(X) > \delta(x)|H_0)$ is commonly referred to as p-value(x).

The bootstrap procedure estimates p-value(x) by drawing a large number of samples from the data set x. These samples are also referred to as bootstrap samples and are created by randomly sampling with replacement from the data set x. The procedure is shown in Algorithm 2, where we reproduce the formulation from Berg-Kirkpatrick et al. [2012].

Algorithm 2 The bootstrap procedure

1. Draw b bootstrap samples $x^{(i)}$ of size n by sampling with replacement from x.
2. Initialize $s = 0$.
3. For each $x^{(i)}$ increment s if $\delta(x^{(i)}) > 2\delta(x)$.
4. Estimate p-value(x) $\approx \frac{s}{b}$.

For a large value of b, the bootstrap estimate of p-value(x) will stabilize. Based on the experiments in Berg-Kirkpatrick et al. [2012], we use $b = 10^6$, which is sufficient for the value to stabilize. For each of the b random samples, the F_1 score of system A as well as system B has to be determined. To make this computation efficient, *TP*, *FP* and *FN* are precomputed for every sentence in the data set. Sampling is then performed over the set of sentences together with the precomputed counts for each sentence. The final F_1 score can be computed from the sum of the precomputed counts for each sentences in the sample.

CHAPTER IV

Part-of-Speech Tagging of Noisy Content

For most parsing models, part-of-speech tagging is an important part of the parsing process. The accuracy of the part-of-speech tagging step has been shown to suffer significantly from noisy content (e.g. Foster et al. [2011]); hence, we will discuss approaches to adapting part-of-speech taggers and how these approaches can be combined in this chapter.

4.1 Domain-specific part-of-speech tagging

Gimpel et al. [2011] present a part-of-speech tagger using a coarse part-of-speech tagset of 25 tags that was specifically designed for and trained on Twitter data. The tagset includes standard part-of-speech tags for nouns, verbs, etc., as well as tags specifically designed to cover tokens mostly seen in social media services, such as URLs, email addresses, emoticons, Twitter hashtags and mentions of usernames.

The part-of-speech tagger is an implementation of a conditional random field with local features. The model uses base features that are commonly used in part-of-speech tagging, such as features for the word type, included digits, suffixes, prefixes, and capitalization. Additionally, several sets of domain-specific features are used. Twitter-specific orthographic features capture expressions such as hashtags or URLs. A gazetteer is used to help detect names since names are often incorrectly capitalized in user generated content. Additionally, distributional similarity features are included to reduce sparseness and the metaphone algorithm is used to produce phonetic normalizations of the input tokens.

In later versions of the part-of-speech tagger [Owoputi et al., 2013], performance was improved by the addition of unsupervised word cluster information acquired via mutual information-based word clustering [Brown et al., 1992]. Word clusters are produced on large amounts of additional unlabeled data. As the word clustering algorithm produces hierarchical classes forming a binary tree, the path to each class can be represented as a bit string in which every bit describes a branching in the tree. Prefixes of this bit string are a representation of the levels of granularity in the word

class hierarchy and are, therefore, used as features in the tagger model. Owoputi et al. [2013] report that the word class features alone are exceedingly efficient in their experiments: a tagger trained with only word class features and transition features already outperforms the tagger presented in Gimpel et al. [2011].

4.2 Combining multiple part-of-speech taggers

As the part-of-speech tagset used by Owoputi et al. [2013] is too coarse to be used by a dependency parser, we want to be able to combine it with part-of-speech tags produced by a part-of-speech tagger with a less coarse tagset.

4.2.1 Universal part-of-speech tagset

Petrov et al. [2011] introduce a coarse part-of-speech tagset of 12 universal part-of-speech categories in order to standardize the part-of-speech tags used in unsupervised and supervised methods. To encourage its adoption, the authors provide mappings between this set and the most common part-of-speech tagsets for various languages. While the tagset itself is too coarse to be used in parsing directly, it is useful for creating a mapping between coarse-grained and fine-grained part-of-speech tags from the tagsets used by various part-of-speech taggers.

4.2.2 N-best part-of-speech tagging

To better combine coarse and fine-grained part-of-speech tags, we determine n-best part-of-speech tags for each token in the sentence. In a hidden Markov model, the probability for all tags occurring at a given position can be calculated using forward and backward probabilities. Using the probability for a tag occurring at a certain position allows the tags predicted for a token to be compared directly. The probability of tag t being the correct tag at position i is defined as:

$$\mathrm{P}(t_i = t) = \alpha_i(t)\beta_i(t)$$

In this formula, α and β are the forward and backward probabilities that can be efficiently computed using dynamic programming. $\alpha_i(t)$ is the total probability of all possible tag sequences ending in the tag t at the ith token and $\beta_i(t)$ is the total probability of all tag sequences starting from t at the ith token and continuing to the end of the sentence [Jurafsky and Martin, 2000, Prins, 2005, p. 65].

4.3 Evaluation

In the experiments we perform, we use a number of combinations of part-of-speech taggers, which we will briefly describe here.

4.3.1 Implementations

We utilize various part-of-speech tagger implementations: For fine-grained tagging, we use the Stanford tagger [Toutanova et al., 2003], which is an implementation of a maximum entropy Markov model, the OpenNLP maximum entropy tagger[1] and an implementation[2] of a trigram HMM tagger [Brants, 2000]. For coarse-grained tagging, we use the domain-specific tagger introduced earlier.

First-best fine-grained tags

In the most basic setup, we use a fine-grained part-of-speech tagger and create coarse part-of-speech tags by mapping the fine-grained tag to the universal part-of-speech tagset.

Independent tags: Twitter-specific coarse-grained and general fine-grained tags

In this setup, we use part-of-speech tags from the Stanford part-of-speech tagger as fine-grained POS tags and a mapping of the Twitter specific POS tags from Owoputi et al. [2013] to the universal tagset as coarse tags.

Twitter-specific coarse-grained and most likely corresponding fine-grained tag

Given the coarse-grained Twitter-specific part-of-speech tagger and a fine-grained general part-of-speech tagger, we aim to jointly tag the text and select the optimal fine-grained part-of-speech tag given the coarse-grained part-of-speech tag. The top k fine-grained part-of-speech tags are determined, and the highest ranking fine-grained part-of-speech tag that is mappable via the universal part-of-speech tagset to the coarse-grained part-of-speech tag of the Twitter-specific tagger is selected. The n-best tag candidates are determined by various methods.

Firstly, in the **n-best sequences** case, we use the OpenNLP maximum entropy tagger to determine n-best tag sequences for the full sentence and then transpose the set of sequences to obtain a ranking for each token.

[1] `http://opennlp.apache.org/`
[2] `https://github.com/danieldk/jitar`

Secondly, in the **n-best tags** case, we use a HMM-based part-of-speech tagger and calculate a ranked list of tags for each token using the forward-backward algorithm introduced above.

Finally, as fast baselines, we determine n-best tags for each individual token. In this case, which is presented under the label **unigram**, the part-of-speech tagger considers only the current token for generating an n-best list of tags. This is a fast but less accurate method since it does not take tag history into account. We use two implementations: The generative implementation uses the emission probability of an HMM tagger $\mathrm{P}(w_i|t_i)$, i.e. the probability of emitting a word w given the tag t at position i, for known words and the probability of generating a word suffix s_i $\mathrm{P}(s_i|t_i)$ in the case of unknown words. The second implementation is a discriminative implementation based on a maximum entropy part-of-speech tagger, which is used to score the current token without tag history features.

4.3.2 Results and discussion

To provide an overview of the performance of the various part-of-speech tagger implementations on noisy content, we parse the dependency test set of Foster et al. [2011], as well as our own dependency parsing test set introduced in Chapter 3. We use a standard MST parser setup.[3] Setups using the Twitter-specific tagger in conjunction with a fine-grained tagger are listed under the label *Joint* below. Results are presented in Table 4.1 and Table 4.2.

Method	System	Unlabeled F_1	Labeled F_1
First-best	Stanford MEMM	73.65	58.14
	OpenNLP ME	72.35	56.47
	OpenNLP MEMM	73.72	57.27
	HMM	73.65	58.17
	Unigram, generative	67.34	51.53
	Unigram, discriminative	71.88	54.20
Joint	Stanford MEMM	73.21	57.50
	OpenNLP ME (n-best sequences)	72.61	57.23
	OpenNLP MEMM (n-best tags)	72.54	56.90
	HMM	72.91	57.17
	Unigram, generative	71.11	55.30
	Unigram, discriminative	72.04	56.20

Table 4.1: Influence of POS tagging on Foster et al. [2011] Twitter development set

[3] An implementation of the MST parser in the Scala programming language is used (`https://github.com/travisbrown/mstparser`).

Method	System	Unlabeled F_1	Labeled F_1
First-best	Stanford MEMM	68.49	56.77
	OpenNLP ME	70.18	58.47
	OpenNLP MEMM	68.60	56.47
	HMM	69.92	57.60
	Unigram, generative	64.19	52.33
	Unigram, discriminative	70.18	58.47
Joint	Stanford MEMM	72.18 S	59.11 S
	OpenNLP ME (n-best sequences)	72.48 S	60.35 S
	OpenNLP MEMM (n-best tags)	72.41 S	60.16 S
	HMM	71.39 S	58.39 S
	Unigram, generative	69.54 Ś	57.64 Ś
	Unigram, discriminative	69.84 Ś	57.07 Ś

Note: S indicates statistical significances at p-value(x) < 0.05, while Ś indicates that the result is not statistically significant.

Table 4.2: Influence of POS tagging on the development section of our test set

Table 4.1 and Table 4.2 show a clear difference between the Foster et al. [2011] test set and our own test set. While in the Foster et al. [2011] test set, the addition of the Twitter-specific part-of-speech tagger degrades parsing performance in all cases (see Table 4.1), the addition of this domain-specific tagger to our own development set improves the parsing performance (see Table 4.2). This result is likely due to the different make up of both test sets, as was discussed in Chapter 3: Because of the method of data collection, the Foster et al. [2011] test set contains cleaner data than our own test set and than the domain for which the Twitter-specific part-of-speech tagger was optimized.

CHAPTER V

Text-Normalization in Parsing of Noisy Content

In this chapter, we assess the influence of text-normalization on dependency parsing of noisy content. In Section 5.1, we introduce and evaluate unsupervised models for text-normalization focused on Twitter data. Section 5.2 discusses text-normalization as a machine translation task and relevant corpora are introduced and evaluated. The treatment of Twitter-specific syntax is discussed in Section 5.3. Finally, in Section 6.1, we provide an overview of the results and perform a detailed error analysis. A discussion of the results follows in Section 6.2.

5.1 Unsupervised lexical normalization

In order to explore unsupervised approaches to text-normalization, we implement a state-of-the-art model for unsupervised lexical normalization of text messages such as Twitter posts and apply it to a Twitter dependency parsing task.

5.1.1 Automatic lexical normalization

Han and Baldwin [2011] treat what they call the message normalization task in analogy to the spell checking task. However, they stress that it differs in that ill-formedness in such a setting is often intentional to a certain degree, for example because there is a message size limit.

The model proposed in Han and Baldwin [2011] operates on tokens and proceeds in two steps: if a token is an out-of-vocabulary word (*OOV*), a classifier is used to determine if the token is a genuine out-of-vocabulary word (e.g. an unknown name) or if it is a possible noisy representation of an in-vocabulary word (*IV*, such as a misspelling or abbreviation).

In the second step, possible in-vocabulary candidates for the OOV token are produced and ranked. The set of initial candidates is the set of in-vocabulary tokens which are within an edit distance of T_c of the OOV token or whose phonetic repre-

sentation (using the double metaphone algorithm) is within an edit distance of T_p of the phonetic representation of the OOV token.

Generating in-vocabulary token candidates

In our implementation, we generate IV token candidates within surface edit distance T_c and within phonetic edit distance T_p of the OOV token by using a Levenshtein automaton [Mihov and Schulz, 2004]. We use the *Infinite Automata* data structure library,[1] which implements Levensthein automata based on Mitankin [2005]. A Levenshtein automaton is a data structure that can be used to solve the problem of fast approximate string matching in dictionaries:

> Given a pattern P, a dictionary D, and a small bound k, efficiently compute the set of all entries W in D such that the Levenshtein distance between P and W does not exceed k. [Mihov and Schulz, 2004]

A universal deterministic Levenshtein automaton is used in conjunction with a deterministic finite state automaton representing the dictionary to solve this problem. In our case, the dictionary is the set of in-vocabulary words. Candidate words in the dictionary automaton are determined by searching within the dictionary automaton and then simultaneously backtracking in both automata. Using this method, it is possible to retrieve candidate corrections for an OOV token in $\mathrm{O}(n)$, where n is the length of the input token [Mihov and Schulz, 2004].

Parameters and features

The maximum surface edit distance T_c and phonetic edit distance T_p are adopted from Han and Baldwin [2011], where they were empirically set to 2.0 and 1.0 respectively. All in-vocabulary token candidates are ranked by an unweighted linear combination of the following features:

- normalized edit distance
- normalized edit distance of the double metaphone representation
- longest common substring
- Is the OOV token a subsequence abbreviation of the IV token (e.g. *ppl* for *people*)?
- Is the OOV token a possible prefix of the IV token?

[1] `http://www.infiauto.com/projects/datastr/`

– Is the OOV token a possible suffix of the IV token?

– normalized score from the dependency-based bag of word model

– normalized score from a trigram language model trained on the subset of text from the target domain containing only IV tokens

The features used in this model measure how close the OOV token is to its possible IV correction (edit distance, longest common substring) and whether it was created through a word formation process (subsequence abbreviation, prefix, suffix). The dependency-based bag-of-words model and the language model further attempt to disambiguate the token based on its surrounding context. The language model used in our experiments was created from a subset of a large collection of tweets, and the dependency-based bag-of-words model was created from the dependency parse trees generated by the Stanford parser for the full content of the English Wikipedia[2].

Examples

Since this model performs normalization only on the level of individual tokens, the number of tokens in each normalized sentence remains constant. As an illustration of the output of the model, consider the following first-best normalization examples of sentences in the Twitter development set of Foster et al. [2011]. In the examples shown in (1), (2) and (3), the original sentence is shown in a. and the proposed correction is presented in b. Tokens that were classified as misspellings are highlighted.

(1) a. **jus** ordered my **terrell owens** jersey
b. **just** ordered my **Terrell owns** jersey

(2) a. Successful businesses benchmark **themsleves** against others .
b. Successful businesses benchmark **themselves** against others .

(3) a. **mabey** we 'll catch watchmen !
b. **maybe** we 'll catch watchmen !

5.1.2 Dictionary-based lexical normalization

In Han et al. [2012], the authors further simplify their previous model. While the model from Han and Baldwin [2011], which was presented above, searches for correction candidates for each OOV token at runtime, the approach presented in Han

[2] `http://www.let.rug.nl/gosse/Wikipedia/enwiki.html`

et al. [2012] works by creating a dictionary of OOV tokens and their IV replacement. The dictionary is applied by replacing each occurrence of an OOV dictionary entry with its IV correction. The manual collection of such a dictionary would be costly; hence, it is constructed in a similar fashion as in the previous model.

One apparent shortcoming of this approach is that there is no disambiguation, i.e. each occurrence of a specific token is always replaced with the same correction. This can be problematic in ambiguous cases; for example, the OOV token *y* can both stand for the IV token *why* and the IV token *you*. The authors present the two example contexts *yeah, y r right!* and *AM CONFUSED!!! y you did that?* for the two readings. This issue is addressed by only including OOV tokens longer than a specified minimum character threshold, which the authors consider being less likely to be ambiguous.

Several individual normalization dictionaries, as well as combinations of individual dictionaries are evaluated. The dictionaries consist of both hand-built dictionaries, which were compiled manually from sources such as the Urban Dictionary,[3] and automatically generated dictionaries.

The automatically constructed dictionaries are created as follows. For a large corpus of English tweets, each OOV token is considered. The dictionary is then extracted in two steps:

1. Extract (OOV, IV) candidate pairs based on distributional similarity.
2. Re-rank the extracted pairs by string similarity.

The result of the algorithm is a list of (OOV, IV) pairs ordered by their string similarity. The best n pairs are then finally included in the normalization lexicon.

The first step exploits the idea that what the authors call *lexical variants* of a token will occur in the same contexts as the token itself. Therefore, given sufficient data, such lexical variants should be observed a number of times in distributionally similar contexts to their standard forms. To measure the distributional similarity, the authors evaluate various features and finally employ context similarity based on tokens in a context window of ± 2 tokens, token bigrams, the position of the token in the sentence and the Kullback-Leibler divergence as an information theoretic distance measure.

The distributionally similar (lexical variant, standard form) word pairs will, however, contain a number of false positive correction pairs such as (*Mnday*, *Tuesday*), with the false standard form *Tuesday* for the misspelling of *Monday*, which is a candidate due to the close distributional similarity of names of weekdays. For this reason, the distributionally similar pairs from the first step are in the second step re-ranked

[3] `http://www.urbandictionary.com/`

by string similarity. Performing this process on an input corpus of appropriate size is a resource-intensive task; however, once the dictionary is built, it can be used without the requiring any further computation.

For re-ranking the extracted pairs, a number of string similarity measures are evaluated based on existing data. The best relative results are achieved using a string subsequence kernel [Lodhi et al., 2002]. For two input strings, a string subsequence kernel provides a measure for the number of common subsequences of a given length (n) between the two strings. Since the computation of overlapping string subsequences is expensive, the authors use a small value for the length of each subsequence ($n = 2$).

Overall, good lexical normalization results are achieved for the combination of a hand-built dictionary and a dictionary automatically produced using the string subsequence kernel for re-ranking. The addition of a dictionary from previous research resulted in the three combined dictionaries providing the best performance in the evaluation.

In the experiments we perform, we will only use the final normalization dictionary that resulted from this research.

5.1.3 Lexical normalization in dependency parsing of Twitter data

To evaluate the impact of text-normalization systems on dependency parsing, we evaluate an English dependency parser on the dependency test set of Foster et al. [2011], which is a set of automatically generated and manually corrected phrase-structure parse trees for Twitter data, which was automatically converted to dependency format.[4]

Experiment

For this experiment, we use the MST parser [McDonald et al., 2005] trained on sections 2–21 of the Penn Treebank. Our only modification is that while the original parser implementation uses only the first character of each fine-grained part-of-speech tag as coarse-grained part-of-speech tags, we use the mapped coarse-grained universal part-of-speech tagset from Petrov et al. [2011].

Test set	Unlabeled F_1	Labeled F_1
WSJ section 23	91.8	89.2

Table 5.1: Baseline parser performance with gold POS tags

[4] `http://nlp.cs.lth.se/software/treebank_converter/`

On section 23 of the Penn treebank, the default setup with second-order features and using gold part-of-speech tags achieves the performance shown in Table 5.1. We use this parser without any modification on the development set of the Twitter data from Foster et al. [2011]. In the configurations in which we do not use gold tags, we use the Stanford part-of-speech tagger to provide part-of-speech tags. For the tests with the dependency test set introduced in Chapter 3, we use the part-of-speech tagger combinations with the Twitter-specific part-of-speech tagger described in Chapter 4.

5.1.4 Results and discussion

In order to determine the impact of lexical normalization on parser performance, we perform the following experiment. First, we determine the performance of the vanilla system using predicted tags and gold tags. The results are presented in Table 5.2.

POS tags	**System**	**Unlabeled F_1**	**Labeled F_1**
Predicted	Vanilla MST parser	73.65	58.14
Gold	Vanilla MST parser	80.42	66.34

Table 5.2: Baseline parser performance on the Foster et al. [2011] development set

Next, we determine the performance of lexical normalization settings, which we will briefly introduce here. In all settings, the tokens of the data set are first normalized before the parser is applied. However, we test various setups to approximate upper-bounds of the performance of lexical normalization on the data set. For each token in the sentence that is corrected by the system, there are usually a number of possible corrections, ranked by their score for the specific token and context. Beyond only the best-ranking normalization suggestion, we are interested in how the n-best performance of the normalization system. For example, while the correct normalization might not be the best-ranked normalization for an OOV token, it might be among the top-ranked suggestions.

The following setups are evaluated:

- In the **first-best normalization** setup, we perform first-best lexical normalization on the input tokens. This means that for each OOV to be corrected, we correct it with the best IV token candidate. The normalized resulting sentences are parsed using the vanilla MST parser model.
- In the **first-best by parse score** setup, we perform n-best lexical normalization, then parse all possible combinations of the n-best lexical normalizations for all tokens and select the parse with the highest parse score.

– The **first-best by parse and normalization score** setup performs n-best lexical normalization. We then parse all possible combinations of lexical normalizations and select the parse $\hat{p}$ with parse score S_P and normalization score S_N, such that $\hat{p} = \arg\max_p S_P(p)\ S_N(p)$.

– The **oracle n-best** setup is used to determine an upper bound for the performance of the text-normalization system: for each OOV word classified as ill-formed, we generate the n-best normalized tokens ($n = 10$). All possible combinations of normalized tokens in the sentence are produced, tagged and parsed. When evaluating, we choose the combination of normalizations such that the parse of the sentence has the highest unlabeled accuracy according to the gold standard. Since this setup relies on the gold standard tree, this setup is, of course, not usable outside the evaluation setting. However, it is useful to indicate how the system would perform if it would always make the correct choice in normalizing a token.

Table 5.4 shows the results of the dependency parsing task after applying lexical normalization.

POS tags	System	Unlabeled F_1	Labeled F_1
Gold	First-best by parse score	80.38	66.31
	First-best by parse+norm. score	80.38	66.31
Predicted	Vanilla MST parser	73.65	58.14
	First-best by parse score	72.74	57.57
	First-best by parse+norm. score	72.74	57.57
	First-best normalization	72.91	57.57
	Dictionary-based normalization	73.55	58.17 $\not\$$

Note: $\$$ indicates statistical significances at p-value(x) < 0.05, while $\not\$$ indicates that the result is not statistically significant.

Table 5.3: Influence of unsupervised lexical preprocessing on the Foster et al. [2011] development set

The results show that the first-best normalization, using the Han and Baldwin [2011] system in either one of its three configurations, degrades the parsing performance of the vanilla MST parser on this data set. For the dictionary-based normalization, there is a slight increase in labeled F_1 score; however, the increase is not statistically significant ($\not\$$ indicates that for this result, p-value(x) $\not< 0.05$).

Given these results, we ran an additional experiment to determine the influence of the detection of ill-formed tokens in the normalization model. For this, the normlization system provides two candidate corrections for each each normalized token:

the first-best normalized token and the original token. In the final parse, out of the original token and the correction, the one token that produces a higher unlabeled accuracy is selected. This simulates the case that the ill-formed word detector is always correct when asserting a token to be ill-formed.

POS tags	System	Unlabeled F_1	Labeled F_1
Predicted	Vanilla MST parser	73.65	58.14
	First-best normalization	74.01	58.40

Table 5.4: Influence of unsupervised lexical preprocessing on the Foster et al. [2011] development set given perfect ill-formed word detection

The performance of the oracle n-best setups is shown in Table 5.5. The results for these setups show that given that the system always selects the best possible normalization (with different sizes n of the set of best candidates), there would be an improvement on this data set. Since for the data set introduced in Chapter 3, gold normalizations are available, we will be able to provide actual upper bounds for the influence of text-normalization on this data set.

POS tags	System	Unlabeled F_1	Labeled F_1
Gold	Oracle n-best (n=2)	80.48	66.37
	Oracle n-best (n=5)	80.52	66.44
	Oracle n-best (n=10)	80.52	66.44
Predicted	Oracle n-best (n=2)	74.08	58.63
	Oracle n-best (n=5)	74.38	58.80
	Oracle n-best (n=10)	74.64	58.80

Table 5.5: Oracle normalization parser performance on the Foster et al. [2011] development set

Overall, perfect ill-formed word detection and improved candidate selection in the Han and Baldwin [2011] model could increase the parsing performance on the Foster et al. [2011] test set; however, the default non-oracle settings for the text-normalization system degrade the parsing performance. This result is similar to the results for part-of-speech tagging we have presented in the previous chapter. Hence, we will next turn to the effect on our own test set.

Table 5.6 presents the results for unsupervised lexical normalization on the development section of the test set introduced previously. Results are shown for two standard part-of-speech taggers (the Stanford and the OpenNLP ME taggers) and the combination of a maximum entropy Markov model for fine-grained tagging and the coarse-grained domain-specific part-of-speech tagger.

POS tags	System	Unlabeled F_1	Labeled F_1
Stanford ME	Vanilla MST parser	68.49	56.77
	First-best normalization	68.53 $\not\text{S}$	56.85 $\not\text{S}$
	Dictionary-based normalization	70.11 S	58.28 S
OpenNLP ME	Vanilla MST parser	70.18	58.47
	First-best normalization	69.81	58.05
	Dictionary-based normalization	70.82 S	59.26 S
coarse+n-best tags	Vanilla MST parser	72.41	60.16
	First-best normalization	71.95	59.71
	Dictionary-based normalization	72.71 S	60.61 S

Note: S indicates statistical significances at p-value(x) < 0.05, while $\not\text{S}$ indicates that the result is not statistically significant.

Table 5.6: Influence of unsupervised lexical preprocessing on the development section of our test set

Only in the case of the Stanford part-of-speech tagger, an improvement for the Han and Baldwin [2011] normalization system can be observed; however, on this data set, the improvement is not statistically significant. The dictionary-based approach from Han et al. [2012], however, shows statistically significant improvements in both unlabeled and labeled F_1 score, and in combination with all part-of-speech tagger setups. A detailed discussion of these results will follow in Section 6.2.

5.2 Text-normalization as machine translation

In this section, we will motivate the idea of using a statistical machine translation system as a preprocessing step for text-normalization and will briefly introduce the basics of statistical machine translation based on phrase-based models as well as discuss corpora relevant to the problem.

5.2.1 Statistical machine translation using phrase-based models

Research in machine translation has a long history spanning over 60 years, and various approaches as well as a variety of levels of encoded linguistic knowledge have been utilized. In recent years, phrase-based systems have provided good results for many languages.

The main idea behind phrase-based models for statistical machine translation is that not words or syntactic structures but phrases form the basic units for translation. A phrase in this sense is any sequence of tokens of a particular length; hence, the

notion of a phrase in this context is not necessarily linguistically motivated.

A phrase-based machine translation system translates a sentence from a foreign language (e.g. French) into a second language (e.g. English) by first segmenting the French sentence into phrases, translating the phrases individually into English and reordering them according to the target language. For the translation of the individual phrases, a translation table is needed, which can be acquired from parallel data in various ways.

Acquiring the translation table

The first step for most algorithms is the acquisition of word alignments. Word alignments are the automatically determined alignments between the words of the sentence and its translation. In the word alignments, words from one language may be aligned to one, many or no word in the target language.

The most widely used word alignment algorithms are based on the IBM Models, which are generative models for word-based translation. The most basic IBM Model, IBM Model 1, defines the translation probability of a foreign sentence to a target-language sentence as follows (we use the formulation from [Koehn, 2010, p. 86]). Let $\mathbf{f} = (f_1, ..., f_{l_f})$ be a foreign sentence of length l_f, let $\mathbf{e} = (e_1, ..., e_{l_e})$ be a target-language sentence of length l_e and let $a : j \rightarrow i$ be an alignment function mapping each target-language word e_j to a foreign word f_i. Then, the probability of translating a foreign sentence **f** to a target-language sentence **e** (in machine translation literature, the target-language is often assumed to be English, hence **e**) is:

$$p(\mathbf{e}, a|\mathbf{f}) = \frac{\epsilon}{(l_f + 1)^{l_e}} \prod_{j=1}^{l_e} t(e_j|f_{a(j)}) \tag{5.1}$$

In this model, $\frac{\epsilon}{(l_f+1)^{l_e}}$ is used for normalization. There are l_f words and one `NULL` token in the foreign sentence ($l_f + 1$), which can be aligned to any of the l_e words in the target-language sentence (hence $(l_f + 1)^{l_e}$). ϵ is a normalization constant. The important part of the formula is the product over the individual lexical translation probabilities for the generated words in the target-language.

To learn the translation probabilities $t(e_j|f_{a(j)})$, a version of the Expectation Maximization (EM) algorithm is applied to sentence-aligned parallel text. While it would be more straight-forward to learn these probability distributions from manually word-aligned sentences, the availability of such corpora would be an unrealistic requirement and sentence-aligned parallel texts can be acquired more easily than word-aligned corpora, for example from the parallel proceedings of the European parlia-

ment.[5]

The EM algorithm is executed with initial probability distributions, which it refines iteratively using the following steps:

1. Initialize the model, for example with uniform or random distributions
2. Expectation step: Apply the model to the data
3. Maximization step: Re-estimate the model from the data
4. Repeat from 2 until convergence

In the case of word alignments, uniformly initialized probability distributions mean that every foreign word may be translated with any target-language word with equal probability. The expectation and maximization step are repeated until the model converges. It has been shown that in some restricted cases, such as the IBM model 1, convergence to a global minimum is guaranteed. Higher IBM models offer refinements on the basic model presented above but also increase the complexity. IBM model 2 adds absolute alignment, IBM model 3 adds a fertility model, IBM model 4 adds a relative alignment model and IBM model 5 corrects deficiencies in the lower-level models. More details on the implementation of the EM algorithm in the case of the IBM models can be found in e.g. [Koehn, 2010, p. 89]). Word alignment based on IBM models 1–5, as well as an HMM-based word-alignment model are implemented in the widely-used GIZA++ toolkit [Och and Ney, 2003], which is also used by the Moses system [Koehn et al., 2007].

From word alignments to phrase alignments

After word alignments are extracted for the aligned parallel sentences, the goal is to find phrases that are consistent with the word alignments. Algorithms for phrase extraction use the alignment matrix as input and aim to find the optimal phrases given the constraints of the word alignment and additional considerations, such as that a target-language phrase that does not contain aligned words should not be matched against the foreign sentence.

The phrase-based model

The phrase-based model for statistical machine translation is defined as follows. We use the formulation from [Koehn, 2010, p. 129]. The best target-language translation

[5] `http://www.statmt.org/europarl/`

$\mathbf{e}_{\text{best}}$ for a foreign sentence $\mathbf{f}$ is

$$\mathbf{e}_{\text{best}} = \arg\max_{\mathbf{e}} p(\mathbf{e}|\mathbf{f}) \tag{5.2}$$

and after applying Bayes' rule:

$$\mathbf{e}_{\text{best}} = \arg\max_{\mathbf{e}} p(\mathbf{f}|\mathbf{e}) p_{\text{LM}}(\mathbf{e}) \tag{5.3}$$

In this model, $p(\mathbf{f}|\mathbf{e})$ is decomposed into:

$$p(f_1^I|e_1^I) = \prod_{i=1}^{I} \phi(f_i|e_i) d(\text{start}_i - \text{end}_{i-1} - 1) \tag{5.4}$$

Using this decomposition, the foreign sentence $\mathbf{f}$ is modeled as consisting of I phrases f_i. $\phi(f_i|e_i)$ is the translation probability of translating a target-language phrase e_i to a foreign phrase f_i.

The reordering model $d(\text{start}_i - \text{end}_{i-1} - 1)$ is a simple distance-based reordering model. The reordering distance $\text{start}_i - \text{end}_{i-1} - 1$ is the number of skipped words between the translation of the ith target-language phrase and the translation of the $i-1$th target-language phrase. Instead of estimating the probabilities for reordering from data, phrase-based MT systems often use an exponential decaying cost function for d. In this case, d is defined as $d(x) = \alpha^{|x|}$, where α is an appropriately estimated parameter between 0 and 1. This formulation of the reordering model penalizes the movement of phrases over long distances and favors short or zero movements.

Extensions of the basic model

The presented model for phrase-based machine translation is the most basic model, which is further refined in most cases. One common refinement, which is, for example, used in the Moses toolkit, is the usage of log-linear models. The basic model assumes that the translation model, the language model and the reordering model all contribute equally to the probability of the final best translation. However, in many cases this assumption is not sufficient, for example, since the data used for the language model may be less reliable than the translation model data. In order to introduce weights into the model, the model can be formulated as a log-linear model,

which is defined as follows.

$$p(x) = \exp \sum_{i=1}^{n} \lambda_i h_i(x) \tag{5.5}$$

h_i is a feature function that is given the full random variable x as input. n is the number of total feature functions used in the log-linear model. Every feature function h_i is weighted by a corresponding weight λ_i. In the basic case, there are 3 feature functions:

- the translation model $\log \phi$,
- the language model $\log p_{\text{LM}}$,
- and the reordering model $\log d$.

Apart from the assignment of weights to its individual components, this model also enables the simple addition of further scores.

Additional common refinements to the model are the usage of bi-directional translation probabilities, word and phrase penalties, discriminative training and extended reordering models. Further, we do not discuss decoding here, which is the essential task of finding the best-scoring translation hypotheses according to the model in a tractable manner.

5.2.2 Corpora

As discussed in the previous section, a phrase-based statistical machine translation system requires sentence-aligned parallel texts to estimate the translation model, as well as monolingual text in the target-language to estimate the language model.

In this thesis, we focus on noisy content from the Twitter domain. For this domain, no useable parallel data of reasonable size exists. There are, however, parallel texts for the text messages domain. These corpora consist of text messages and their normalized equivalents. Such corpora exist since there has been a growing interest in the automated processing of text messages, for example for supporting the work of emergency responders in natural disasters such as earth quakes. While this is a different domain from the domain we focus on, the restrictions on both domains are similar. Similarly to the domain we focus on, text messages are restricted to a fixed number of characters, hence inviting users of the medium to abbreviate words in order to maximize the content of the message. Additionally, as with the Twitter domain, there is often little focus on correct spelling and messages are often composed in a

very limited amount of time. Therefore, these data sets should be applicable for our purposes. We will briefly introduce the relevant corpora in this section.

Parallel text message corpora

Aw et al. [2006] explore phrase-based statistical machine translation as a preprocessing step to a machine translation task involving text messages. As part of their effort, they manually normalize a subset of 5.000 messages from a larger corpus of text messages.

Raghunathan and Krawczyk [2009] explore the usage of standard toolkits for machine translation to perform a similar text-normalization task and extend the corpus created by Aw et al. [2006] by around 2.500 normalized text messages.

Both corpora were not directly created for social media services such as Twitter, which is the source of our data set, however the restrictions of both domains are similar and hence the assumption that we can use both corpora for our purpose is reasonable. Table 5.7 provides an overview of relevant corpus statistics for both the Aw et al. [2006] corpus and the training, development (D) and test (T) sections of the Raghunathan and Krawczyk [2009] corpus. The out-of-vocabulary rate for tokens and types is calculated against the English dictionary of the GNU Aspell spell checker.[6]

		Sentence length			**Token length**			**OOV rate**	
Corpus	# sent.	Mean	Med.	Std	Mean	Med.	Std	Tokens	Types
Aw EN	5000	13.90	12	8.60	3.45	3	1.99	0.327	0.405
Aw SMS	5000	13.73	11	8.51	3.18	3	1.90	0.392	0.500
R&K EN	1930	14.57	13	9.42	3.44	3	1.90	0.230	0.235
R&K SMS	1930	14.26	12	9.18	3.13	3	1.85	0.315	0.327
R&K EN D	540	15.84	13	10.13	3.39	3	1.85	0.239	0.197
R&K SMS D	540	15.44	13	9.77	3.06	3	1.80	0.321	0.308
R&K EN T	477	17.06	15	10.60	3.43	3	1.86	0.246	0.226
R&K SMS T	477	16.52	14	10.13	3.11	3	1.77	0.346	0.353

Table 5.7: Corpus statistics for parallel text message corpora

5.2.3 Methodology

As in the previous section, we perform a simple evaluation of machine translation as a preprocessing step for parsing by applying it to a dependency parsing task.

[6] `http://aspell.net/`

System setup

Based on the noisy parallel texts and following the Moses baseline tutorial,[7] we create a Moses system [Koehn et al., 2007]. The training corpus is formed by concatenating the parallel texts from Aw et al. [2006] and the training section from Raghunathan and Krawczyk [2009]. Word alignments are automatically determined via the GIZA++ toolkit. The `grow-diag-final` heuristic is used to establish the final word alignments based on the two word alignments produced by GIZA++ (the two sets of word alignments are the two directions `original` → `normalized` and `normalized` → `original`). A language model is built using the IRSTLM toolkit[8] on the English side of the *news-commentary* data set used in the baseline setup.

The weights λ for the log-linear model used in Moses are automatically estimated on heldout data using the Minimum Error Rate Training (MERT) algorithm [Och, 2003]. Tuning is performed on the development section of the Raghunathan and Krawczyk [2009] corpus.

Baseline performance

Table 5.8 presents the performance of the machine translation system on the development section of our data set and on the test section of the text message data that was used for training this system. Table 5.8a shows the performance scores for the normalization of the development section of our data set and Table 5.8b shows the results for the test section of the Raghunathan and Krawczyk [2009] data. In both tables, ↑ and ↓ indicate whether a higher score (↑) or a lower score (↓) constitute an improvement. In the two columns **Raw** and **Normalized**, we show the results of translation quality metrics between the original, raw tokens and their gold normalization (**Raw**) and the automatically normalized raw tokens and the gold normalization (**Normalized**). Results are shown using the BLEU metric [Papineni et al., 2002], which is based on overlapping n-grams, and the translation edit rate [Snover et al., 2006], which models the number of edits required to correct the output of a system. Finally, the length of the translation hypotheses as a ratio to the length of the gold normalization is displayed in the table row **Length**.

Due to slightly differing annotation conventions between the data sets, we perform a minimal correction before the evaluation: For the evaluation of translation quality, we use two reference translations, first the gold standard translation and second the gold standard translation with common contractions expanded to their full

[7] `http://www.statmt.org/moses/?n=Moses.Baseline`
[8] `http://hlt.fbk.eu/en/irstlm`

word forms.[9] This expansion is necessary because in the Raghunathan and Krawczyk [2009] training set, contractions are consistently normalized to their full word forms and hence the MT system replicates this behavior. By using both versions as reference translations, we qualify both the contracted versions and the full versions as acceptable in the evaluation. All scores are for fully lowercased, tokenized references and translation hypotheses.

Metric	Raw	Normalized
BLEU ↑	85.0	87.0 **S**
TER ↓	7.8	6.1 **S**
Length	97.9	98.9

(a) Development section of our data set

Metric	Raw	Normalized
BLEU ↑	50.9	84.4 **S**
TER ↓	25.2	7.6 **S**
Length	96.9	99.3

(b) Raghunathan and Krawczyk [2009] test section

Note: **S** indicates statistical significances at p-value(x) < 0.05, while **$\not$S** indicates that the result is not statistically significant.

Table 5.8: Metrics for MT output quality for the normalization of two test data sets

From these baseline results, it is apparent that the normalization is both more urgent and more effective on the Raghunathan and Krawczyk [2009] test section specifically targeted to text message normalization. The BLEU score of the non-normalized text of 50.9 on the text message data set compared to the BLEU score of 85.0 on our Twitter data set shows that this data set requires significantly more normalization. Our data set was collected as a representative sample, and not as a data set containing exclusively noisy data. Nevertheless, the normalization using the Moses baseline system improves the BLEU score on our data set, albeit by a smaller margin than on the text message data. The improvement in both cases is also visible in the length of the automatically normalized sentences, which are closer to the length of the gold normalization in both cases.

5.2.4 Results and discussion

Having introduced and evaluated the Moses baseline translation system, we will investigate the effectiveness of the translation system as a preprocessing step to reduce noise in a dependency parsing task. As previously, we perform a simple parsing task on noisy content with different setups of a dependency parser. In this part, we are only interested in the influence of the machine translation system and will hence only

[9] Specifically, we perform the following replacements: *'ll* → *will*, *'ve* → *have*, *'m* → *am*, *n't* → *not*, *ca* → *can*, *'re* → *are*

compare three setups: as a baseline, we use the MST parser with no normalization. The second setup is the system, where normalization is performed using the Moses setup as a preprocessing step before the parsing task and the third system is a system, where we use the manually normalized tokens from our development set as gold normalization. All parser setups use the Twitter-specific part-of-speech tagger for coarse tagging and n-best part-of-speech tags from a maximum entropy Markov model for fine-grained tagging. Table 5.9 shows the results for each setup in terms of unlabeled and labeled F_1 score, as introduced in Section 3.3.

POS tags	Data set	Normalization	Unlabeled F_1	Labeled F_1
coarse+n-best tags	Dev.	Vanilla	72.41	60.16
		Moses	73.25 $\not$S	61.26 S
		Gold	77.30 S	65.82 S
	Full	Vanilla	71.35	59.32
		Moses	72.25 S	60.38 S
		Gold	76.36 S	64.96 S
	Foster dev.	Vanilla	72.55	56.90
		Moses	72.27	56.50

Note: S indicates statistical significances at p-value(x) < 0.05, while $\not$S indicates that the result is not statistically significant.

Table 5.9: Influence of MT-based preprocessing on three data sets

It can be observed that the application of the Moses system as a preprocessing step improves the unlabeled F_1 score of our dependency parsing task from 72.41 to 73.25 on the development section and from 71.35 to 72.25. Hence, as such, a preprocessing step based on machine translation does improve parsing quality.

On the development section of our data set, the difference between the vanilla system and the system using Moses normalization is not statistically significant (for p-value(x)< 0.05). However, since the significance of the result is influenced by a number of factors, including the size of the data set, we also performed the same evaluation on the combined development and test sections of our data set (*Full* in Table 5.9). On the full data set, the improvement is significant (p-value(x)$= 0.018$).

On the development set of Foster et al. [2011], the normalization degrades the vanilla parsing performance. This is a similar result to the unsupervised methods; however, the this decrease is not as strong as the decrease we have observed there.

An equally interesting result, however, is that the upper bound of the normalization-based improvement, which would assume that a normalization step produces perfect output, is as high as an F_1 score of 77.30. In the next sections, we will perform a de-

tailed error analysis to investigate this observation. Additionally, the normalization in this experiment was applied directly to the full original data, including Twitter-specific syntax and hence we will evaluate the same setup with a separate treatment of Twitter-specific syntax in the next section.

5.3 Twitter-specific preprocessing

In most cases, methods for domain adaptation aim to be generalizable to any number of domains. However, there are also cases in which it is practical to focus on one particular domain. The Twitter-specific part-of-speech tagger introduced in Section 4.1, for example, exhibits a number of properties that apply mainly to its single target-domain, such as specialized part-of-speech tags for Twitter-specific syntax like hashtags. Similarly, we integrate domain-specific treatment of Twitter syntax in our experiments.

In Section 3.2.2, the treatment of Twitter-specific syntax in the dependency annotations for our test set were outlined. In this section, we will discuss the target-domain specific handling of syntax particular to Twitter as part of parsing.

We treat Twitter-specific syntax using the following deterministic procedure:

1. Remove Twitter-specific tokens from the beginning of the sentence and push them on a stack.
2. Remove Twitter-specific tokens from the end of the sentence and push them on a stack.
3. Parse the remaining sentence using the underlying dependency parser.
4. Re-attach the tokens from (1) and (2) to the resulting dependency tree in accordance to the annotation guidelines set out in Section 3.2.2.

Twitter-specific tokens occurring at the beginning of a sentence are discourse markers, such as *RT* and *MT* and user names. Tokens occurring at the end of a sentence are hash tags, URLs and user names.

Table 5.10 shows the influence of the deterministic treatment of Twitter-specific syntax on our dependency test set, as well as on the data set from Foster et al. [2011]. As in previous experiments, the MST parser is used. In this case, the part-of-speech tags are provided by the OpenNLP maximum entropy part-of-speech tagger.

Treating Twitter-specific syntax in the same fashion as it is annotated in our data set leads to a statistically significant improvement in F_1 score over the vanilla system. This result is not surprising since it merely means that the Twitter-specific tokens

POS tags	data set	System	Unlabeled F_1	Labeled F_1
coarse+n-best tags	Dev.	Vanilla	72.41	60.16
		Twitter-specific	76.17 S	64.38 S
Stanford first-best	Foster dev.	Vanilla	73.65	58.14
		Twitter-specific	75.52 S	60.11 S

Note: S indicates statistical significances at p-value(x) < 0.05, while $\not{\text{S}}$ indicates that the result is not statistically significant.

Table 5.10: Influence of deterministic handling of Twitter-specific syntax

that can be unambiguously annotated are annotated automatically in the same way as they are annotated manually in the test data. The same significant improvement can, however, also be observed on the development section of the independent Twitter data set of Foster et al. [2011].

CHAPTER VI

Discussion and Conclusion

In this chapter, we will summarize and discuss the overall results, analyze the errors produced by one system setup on the development section of our data set and discuss possible improvements.

6.1 Evaluation

In the previous sections, we have discussed various methods of preprocessing and part-of-speech tagging of noisy content and have evaluated these methods individually. In this section, we will combine and summarize these methods and perform an error analysis on one of the best-performing combinations.

6.1.1 Summary

The deterministic preprocessing described in Section 5.3 can straightforwardly be combined with other normalization methods. In Table 6.1 we present the results of such combinations. A first division in the table is whether Twitter-specific preprocessing was performed (*Twitter-specific*) or not (*Vanilla*). The table is further subdivided by the method used for normalization and the type of part-of-speech tagger being used. In the case of gold normalization, we directly parse the gold side of our development set. Gold part-of-speech tags are only applicable to the case of gold text-normalization since in other cases the tokens being parsed may not entirely match the part-of-speech gold standard provided by the gold side of the development data set.

The results shown in Table 6.1 allow several observations: While evaluating the performance of MT-based normalization in Section 5.2.4, the difference between MT-based normalization and gold normalization could be observed as rather large (for the case of coarse+n-best tags, F_1 score of 73.25 compared to 77.30). As Table 6.1 shows, after Twitter-specific preprocessing was performed the difference between MT-based and gold normalization is much smaller (F_1 score of 76.85 compared to 78.24).

A second observation is that in all cases, even in the case that Twitter-specific

System	Norm.	POS tags	Unlab. F_1	Labeled F_1
Vanilla	-	Coarse+n-best tags	72.41	60.16
		Coarse+n-best seq.	72.48	60.35
		Fine	70.18	58.47
Twitter-specific	Gold	Gold	79.28	69.85
		Coarse+n-best tags	78.24	67.54
		Coarse+n-best seq.	78.20	68.02
		Fine	77.87	67.35
	Moses	Coarse+n-best tags	76.85	65.38
		Coarse+n-best seq.	**77.04**	**65.64**
		Fine	75.46	64.52
	Norm. dict.	Coarse+n-best tags	76.36	64.80
		Coarse+n-best seq.	76.55	65.17
		Fine	74.25	63.63
	Moses+dict.	Coarse+n-best tags	77.08	65.57
		Coarse+n-best seq.	**77.08**	**65.68**
		Fine	75.58	64.74
	-	Coarse+n-best tags	76.17	64.38
		Coarse+n-best seq.	75.98	64.53
		Fine	73.31	62.61

Table 6.1: Summary of parser performance on the development section of our test set

syntax is handled by preprocessing, combining the coarse-grained part-of-speech tags provided by a Twitter-specific part-of-speech tagger with fine-grained part-of-speech tags provided by a second part-of-speech tagger improves parsing performance by a statistically significant margin.

Finally, there is a minor additional improvement in unlabeled and labeled F_1 score for combining the MT-based normalization with the dictionary-based normalization. However, this difference is not statistically significant.

6.1.2 Error analysis

We perform an error analysis on a selected setup from the last section. This setup uses both Twitter-specific normalization and MT-based normalization, as well as coarse-grained Twitter-specific part-of-speech tagging combined with n-best part-of-speech tags provided by a maximum entropy Markov model.

In order to give an overview of the most common types of errors and their sources, 50 incorrect dependency relations were randomly selected from the development set, which was parsed using the setup introduced above. These dependency relations were

then manually categorized by the type of error that lead to the relation deviating from the gold standard. Table 6.2 shows the resulting manually judged error types arranged into categories and ordered by their frequency within each category.

Category	Error type	Frequency
Domain-specific errors	General errors	7
	Attachment of emoticons	4
	Missing copula	3
Mis-attachment	Attachment of punctuation	6
	Attachment of object	2
	Attachment of subject	1
	Attachment of adverb	1
	PP-attachment	1
Normalization	Unnecessary normalization	2
	Token not split correctly	2
	Mis-normalization	1
	Mis-insertion	1
General	Other errors and inconsistency between training and test data	14
	Proper noun compounds	1

Table 6.2: Error analysis on the development section of our test set

We found four broad categories of errors with several specific error types in each category. The single most frequent error type is "Other errors and inconsistencies between training and test data" followed by the category of errors related to the domain-specific properties of the data set. In the following, the four categories of error sources are defined and illustrated with examples.

Domain-specific errors

Frequent errors in this category are due to dependency relations that are mis-attached because the copula that should be annotated as the root of the dependency tree is missing.

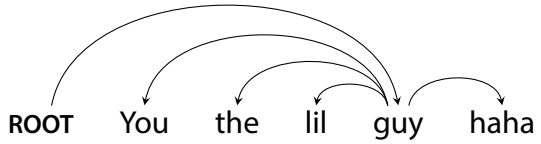


Figure 6.1: Predicted dependency tree missing *are*

In these cases, the normalization failed to insert this token. Figure 6.1 shows the dependency tree predicted for the input tokens *U da lil guy haha*. Since in the gold standard, we assume that a token for *are* was inserted and in this case the dependency parser identified *guy* as the root and attached all other tokens to it, there are three resulting incorrect dependencies in the tree (guy → haha, guy → You and ROOT → guy).

A second frequent source of error in the data set is the treatment of emoticons. In annotating the data set, we reached the decision to treat emoticons like other unclassified utterances such as *haha* and attach them to the root node. As emoticons are not a part of the WSJ training data, the dependency parser treats them as punctuation and attaches them according to the CoNLL conventions for punctuation present in the training data.

Other domain-specific errors (*General errors* in Table 6.2) are mostly due to correctly spelled but domain-specific vocabulary.

Mis-attachments

The category of mis-attachments contains general cases where a specific type of token is not attached to the correct head. These mis-attachments contain linguistically less relevant categories like punctuation, but also the attachment of the object and subject to the correct verb head, the attachment of adverbs and the attachment of prepositional phrases. Unlike the errors discussed in the last category, the errors based on mis-attachment are not very specific to the domain of the data and could be solved by more accurate parsing algorithms.

Normalization-related errors

As normalization-related errors, we treat any instance of an error that is due directly to the preprocessing we performed. An unnecessary normalization is the case that the normalization step replaced a correct token with an incorrect proposed correction.

The error type *token not split correctly* refers to the case that a token in the input data should be split into two or more tokens according to the gold standard but was not split by the preprocessing step. For example, the token *ur* would in a certain context refer to *you are* and should accordingly be split into these two tokens.

Mis-normalization refers to the case that a token that was indeed incorrect was normalized to an incorrect proposed correction.

Finally, the normalization step can also lead to incorrect insertions, which in turn can cause incorrect dependencies that are not part of the gold standard.

General errors

In the category of general errors, we count reasons for incorrect dependencies that are neither specific to the domain nor caused by any normalization or specific linguistic phenomenon, such as errors in the dependencies of parts of compound proper nouns.

The largest number of errors we observed for a single error type were miscellaneous errors and inconsistencies between training and test data. As belonging to this type, we count, for example, cases in which from multiple verbs in a sentence, a different verb than the gold standard root is chosen as the root. This error type contains both cases, were this is due to an inconsistency between the training and our test set annotation and where it is a genuine mistake by the parser.

6.1.3 Possible improvements

The error analysis performed shows that while the normalization appears to work reliably, domain-specific issues remain. Both emoticons and errors due to missing copula form a significant part of the domain-specific errors and could be addressed separately.

General errors not related to the domain or the normalization are either due to genuine ambiguities in the category *mis-attachments* or due to inconsistencies between the training set and the test set. While the domain-specific and normalization-related errors can be addressed by improving the normalization, the errors in the *general* and *mis-attachment* category would also occur on other data sets and could be improved by a better underlying parser model.

6.2 Discussion

The experiments in Section 5.1, Section 5.2, and Section 5.3 examined normalization based on machine translation as well as deterministic preprocessing of Twitter-specific syntax. The findings from these experiments have shown that thse normalization techniques can improve parsing performance on a data set for dependency parsing that is focused on user-generated, noisy content. In Section 5.1, we observed mixed results for unsupervised methods. The results were mixed in the sense that while the purely unsupervised method did not improve performance; the dictionary-based method, which additionally includes manually selected data, did indeed show improvements. The error analysis presented in the previous section has further shown that while a large proportion of the remaining errors are genuine parsing errors, domain-specific errors still remain.

Performance of unsupervised and semi-supervised models

In the experiments, we have observed that the semi-supervised model, which performs text-normalization using a machine translation system, has improved the parsing accuracy on our noisy data set, and slightly degraded the performance on the cleaner data set from Foster et al. [2011]. The unsupervised models, when applied without manually added data, did not improve the performance. On the other hand, the dictionary-based normalization, which contains manually added data, improved the performance. However, this improvement still remains smaller than the improvement obtained from the MT-based normalization system.

At a first glance, this observation seems to be inconsistent with previous evaluations, such as the evaluations performed in the original publications of Han and Baldwin [2011] and Han et al. [2012], in which the unsupervised models have been shown to perform better than models based on statistical machine translation. However, there are several reasons why these observations are compatible with our results:

Firstly, previous evaluations of unsupervised text-normalization methods work only on the token level. This means that even though the models are able to correct single tokens; they are actually unable to split or delete them, or insert new tokens. Returning to a previous example, the token *ur*, which could mean both *your* and *you are* depending on the context, is represented in the Han et al. [2012] normalization dictionary by the entry *your*. In this case, the restriction of the model to single tokens would not allow *ur* to be expanded to *you are*. It could be argued that instead of a part of a normalization component, the normalization of single tokens into multiple tokens should be a part of the tokenization component. However, since there are cases where syntactic disambiguation is necessary, this would merely move the problem to another component, and not provide a useful solution to the normalization problem. Additionally, since insertions and deletions should be possible, this task is better handled in the normalization step. Text-normalization using a machine translation system does not require the restriction to single tokens since in statistical machine translation, each source-language token can be translated to a `NULL` token, a single token or multiple tokens in the target language.

The second reason why these observations do not necessarily contradict our results is because the overall results depend heavily on the make up of the test set and the test methodology. On the Foster et al. [2011] data set, for example, the methods result in only negligible changes. The goal for our data set was to make it as representative as possible for the average language use in Twitter. Therefore, it is still less noisy than test sets, which are more focused on noise. Therefore, the unsuper-

vised model might still perform better on those test sets consisting of even noisier data. Overall, however, it is also encouraging to observe that combining the MT-based normalization with the unsupervised normalization can still improve on the results of both further, even if this difference is not statistically significant.

Alternatives

Performing preprocessing of a parsed text is only one possible means of parser adaptation. For instance, it has been shown that methods such as self-training, i.e. retraining a parser on the in-domain output of a more exact but slower generative phrase structure parser, can provide significant improvements with exceedingly large data sets. However, one advantage of preprocessing as parser adaptation is that it is agnostic to the underlying parser, which offers several benefits. Firstly, it can straightforwardly be directly with other means of adaptation such as self-training. Secondly, in the basic case, existing parsing models can be utilized directly without any modification and, hence, it is straightforward to exchange the underlying parser.

Outlook and future work

In domain adaptation, and natural language processing in general, a common problem is the scarcity of relevant annotated data sets. It is generally assumed that while unannotated text for a domain is relatively unproblematic to obtain; annotated data sets are usually scarce. Self-training is, therefore, a valuable domain adaptation strategy as it only requires unannotated in-domain data (along with a strong generative parser). Normalization based on MT techniques, however, requires parallel texts, which are easier to acquire than syntactically annotated data, but they still pose a minor obstacle.

In our case, we were able to utilize existing data sets for a related task, but for other languages, such data sets might not be available. For these reasons, we also evaluated unsupervised methods for text-normalization in Section 5.1. In our experiments, these methods did not provide the same improvement as the MT-based method we explored, and thus more investigation into this issue may improve this situation.

6.3 Conclusion

In this thesis, we have compared various strategies for adaptation to noise in dependency parsing. The central question of interest was whether a text-normalization step based on MT techniques and whether a text-normalization step using unsupervised methods can be applied efficiently to the parsing task.

In order to examine this issue, a test data set containing noisy content from Twitter was collected and annotated, and an evaluation metric was defined. The data set was created with the goal that it had to be representative both in terms of language use and in terms of the general level of noise within this domain. Since we were interested in working directly with the noisy version of the data, the data set was designed to include both the original data and the normalized version of the text that was annotated with dependency syntax. We hope that beyond this work, this data set will be of utility to related research areas, such as research into human sentence comprehension.

Using this data set, we have shown that text-normalization as a preprocessing step in dependency parsing can lead to significant improvement of parsing accuracy over a baseline system. The experiments demonstrated that text-normalization based on standard machine translation tools, and combined with deterministic treatment of selected domain-specific syntax, provides good results. The machine translation system is trained on parallel texts created for a text message normalization task, which is not strictly the same as our target domain but is a related setting since the restrictions of both domains are similar. Further, we have shown how a domain-specific and a general part-of-speech tagger can be combined for this task.

For someone uninitiated in the inner workings of modern parsing algorithms, a natural language parser's difficulty with even small errors in its input data might be difficult to comprehend. After all, when we, as human beings, read a text containing minor errors, we do not have any difficulty ignoring these errors and understand the intended message regardless. The noisy-channel model, which is the basis for both the MT-based methods and the unsupervised methods for text-normalization we explored, has been used extensively for modeling sources of distortion in various natural language processing tasks. Applying this model to the parsing task by performing preprocessing of the input text, we observed an improvement in parsing accuracy. While this is only a small step towards making parsing as robust to noise as human sentence comprehension, we still consider this as an interesting result.

Bibliography

AiTi Aw, Min Zhang, Juan Xiao, and Jian Su. A phrase-based statistical model for SMS text normalization. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 33–40. Association for Computational Linguistics, 2006.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. An empirical investigation of statistical significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.*, pages 995–1005. Association for Computational Linguistics, 2012.

Adam Bermingham and Alan F. Smeaton. Classifying sentiment in microblogs: is brevity an advantage? In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management.*, pages 1833–1836. ACM, 2010.

Thorsten Brants. TnT: a statistical part-of-speech tagger. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*, ANLC '00, pages 224–231, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

Peter F. Brown, Peter V. Desouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, 1992.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics, 2006.

Yoeng-Jin Chu and Tseng-Hong Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14(1396-1400):270, 1965.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing*, volume 10, pages 1–8, Philadelphia, PA, 2002. Association for Computational Linguistics.

Paul Cook and Suzanne Stevenson. An unsupervised model for text message normalization. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 71–78. Association for Computational Linguistics, 2009.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *The Journal of Machine Learning Research*, 3:951–991, 2003.

Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423. Association for Computational Linguistics, 2004.

Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71:233–240, 1967.

Bradley Efron and Robert Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall, New York, 1993.

Jason M. Eisner. Three new probabilistic models for dependency parsing: An exploration. In *Proceedings of the 16th Conference on Computational Linguistics*, volume 1, pages 340–345. Association for Computational Linguistics, 1996.

Jennifer Foster, Özlem Çetinoglu, Joachim Wagner, Joseph Le Roux, Stephen Hogan, Joakim Nivre, Deirdre Hogan, and Josef van Genabith. #hardtoparse: POS tagging and parsing the twitterverse. In *Analyzing Microtext*, 2011.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, 2011.

Bo Han and Timothy Baldwin. Lexical normalisation of short text messages: Makn sens a #twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 368–378, 2011.

Bo Han, Paul Cook, and Timothy Baldwin. Automatically constructing a normalisation dictionary for microblogs. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning.*, pages 421–432, 2012.

Hideo Hirose and Liangliang Wang. Prediction of infectious disease spread using Twitter: A case of influenza. In *Parallel Architectures, Algorithms and Programming (PAAP), 2012 Fifth International Symposium on*, pages 100–105. IEEE, 2012.

Daniel Jurafsky and James H. Martin. Speech and language processing an introduction to natural language processing, computational linguistics, and speech. 2000.

Max Kaufmann. Syntactic normalization of Twitter messages. In *International Conference on Natural Language Processing*, Kharagpur, India, 2010.

Mark D. Kernighan, Kenneth W. Church, and William A. Gale. A spelling correction program based on a noisy channel model. In *Proceedings of the 13th Conference on Computational Linguistics*, volume 2, pages 205–210. Association for Computational Linguistics, 1990.

Philipp Koehn. *Statistical Machine Translation*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics, 2007.

Sandra Kübler, Ryan McDonald, and Joakim Nivre. *Dependency Parsing*, volume 2 of *Synthesis Lectures on Human Language Technologies*. Morgan & Claypool Publishers, 2009.

Matthew Lease, Eugene Charniak, Mark Johnson, and David McClosky. A look at parsing and its applications. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, pages 1642–1645, No. 2. Menlo Park, CA; Cambridge, MA, 2006. MIT Press.

Roger Levy. A noisy-channel model of rational human sentence comprehension under uncertain input. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 234–243. Association for Computational Linguistics, 2008.

Huma Lodhi, Craig Saunders, John Shawe-Taylor, Nello Cristianini, and Chris Watkins. Text classification using string kernels. *The Journal of Machine Learning Research*, 2:419–444, 2002.

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

P.H. Matthews. Domain. In The Concise Oxford Dictionary of Linguistics., 2013. URL `http://www.oxfordreference.com/view/10.1093/acref/9780199202720.001.0001/acref-9780199202720-e-957`.

David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *Proceedings of the main conference on human language technology conference of the North American Chapter of the Association of Computational Linguistics.*, pages 152–159. Association for Computational Linguistics, 2006.

David McClosky, Eugene Charniak, and Mark Johnson. Automatic domain adaptation for parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 28–36. Association for Computational Linguistics, 2010.

Ryan T. McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajic. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2005.

Stoyan Mihov and Klaus U. Schulz. Fast approximate search in large dictionaries. *Computational Linguistics*, 30(4):451–477, 2004.

Petar Nikolaev Mitankin. Universal Levenshtein automata. Building and properties. Masters thesis, 2005.

Joakim Nivre, Johan Hall, Sandra Kübler, Ryan T. McDonald, Jens Nilsson, Sebastian Riedel, and Deniz Yuret. The CoNLL 2007 shared task on dependency parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL*, pages 915–932, 2007a.

Joakim Nivre, Johan Hall, Jens Nilsson, Atanas Chanev, Gülsen Eryigit, Sandra Kübler, Svetoslav Marinov, and Erwin Marsi. Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95–135, 2007b.

Franz Josef Och. Minimum error rate training in statistical machine translation. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, volume 1, pages 160–167. Association for Computational Linguistics, 2003.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390. Association for Computational Linguistics, 2013.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

Slav Petrov, Pi-Chuan Chang, Michael Ringgaard, and Hiyan Alshawi. Uptraining for accurate deterministic question parsing. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing.*, pages 705–713, 2010.

Slav Petrov, Dipanjan Das, and Ryan T. McDonald. A universal part-of-speech tagset. *arXiv preprint arXiv:1104.2086*, 2011.

Barbara Plank and Gertjan van Noord. Effective measures of domain similarity for parsing. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics*, pages 1566–1576, 2011.

Robbert Paul Prins. *Finite-state pre-processing for natural language analysis.* PhD thesis, University of Groningen, 2005.

Karthik Raghunathan and Stefan Krawczyk. Investigating SMS text normalization using statistical machine translation. 2009.

Frank Rosenblatt. The perceptron. *Psychological Review*, 65(6):386–408, 1958.

Adam Sadilek, Henry A. Kautz, and Vincent Silenzio. Predicting disease transmission from geo-tagged micro-blog data. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, Toronto, Ontario, Canada, 2012. AAAI Press.

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 623–656, July, October 1948.

Libin Shen, Jinxi Xu, and Ralph Weischedel. A new string-to-dependency machine translation algorithm with a target dependency language model. In *Proceedings of*

the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 577–585. Association for Computational Linguistics, 2008.

Matthew Snover, Bonnie Dorr, Richard Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of the Association for Machine Translation in the Americas*, pages 223–231, 2006.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, volume 1, pages 173–180. Association for Computational Linguistics, 2003.

Daniel Zeman and Philip Resnik. Cross-language parser adaptation between related languages. In *Proceedings of The Third International Joint Conference on Natural Language Processing (IJCNLP)*, pages 35–42, Hyderabad, India, 2008.

List of Tables

List of Figures

List of Algorithms