Improving Semantic Frame Accuracy for Semantic Dependency Parsing

Arash Saidi

Thesis submitted for the degree of
Master in Informatics: Language and Communication
60 credits

Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2017

Improving Semantic Frame Accuracy for Semantic Dependency Parsing

Arash Saidi

Improving Semantic Frame Accuracy for Semantic Dependency Parsing

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

This thesis presents an in-depth contrastive error analysis of a set of semantic dependency parsing systems. Based on the empirical results of our analysis we found semantic frame classification to be an interesting case study. As part of this thesis we have made a semantic frame classifier that outperforms previous results. The semantic frame classifier is the result of rigorous experimentation with four set of features: (1) lexical, (2) morphological, (3) syntactic, and (4) semantic. We show that our results outperform previous results. We also show that our classifier can be used to extend and improve the frame semantic classification accuracy of two existing state-of-the-art semantic dependency parsing systems.

Acknowledgements

This thesis is submitted for the degree of Master of Science at the University of Oslo. My supervisors have been Stephan Oepen and Lilja Øvrelid. I am thankful for their insights, helpful advice and patience.

I want to thank my good friend, fellow student, and co-worker Petter Hohle for his encouragement. I thank my girlfriend Marit for everything.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Semantic dependency parsing is a Natural Language Processing (NLP) task that aims at producing meaning representations at the sentence level. Its outcome is to express predicate-argument relations in order to answer the question of *Who did What to Whom?* In this regard there is an overlap between semantic dependency parsing and *semantic role labeling* (SRL).

The task of SRL is concerned with detecting the arguments of a predicate in a given sentence, and is often limited to only verbal predicates. Semantic dependency parsing has a broader scope than argument detection. In addition to the goals of SRL, semantic dependency parsing attempts to identify various semantic phenomena, such as negation, topicalization, relative clauses, and other scopal dependencies that are usually left out in SRL.

An examination of recent research and publications in the field of dependency parsing shows a growing interest in the semantic aspect of dependency parsing. This increasing popularity stems both from advances made in the accuracy of state-of-the-art parsers, successful applications of such parsers and their representations in a wide range of computational tasks, such as Information Extraction, Textual Inference, Machine Translation, Semantic Search, and Sentiment Analysis. A set of so-called *shared tasks* on semantic dependency parsing, which we will present in this thesis, have also stimulated the research community towards more research on this specific type of dependency parsing.

An second focus of this thesis is *semantic frame* classification. A semantic frame is intricately linked to the *sense distinctions* of a word, i.e. how a singular term and its arguments are to be interpreted. This task is related to the NLP task of word sense disambiguation, where different sets of classes are used in order to differentiate words into classes of meaning representation. Where semantic dependency parsing adds a layer of semantics to a sentence by adding connections between words, semantic frames adds a layer by adding several interpretations to individual words.

This thesis aims to contribute to the field of semantic dependency parsing and frame classification by performing a high-level contrastive error analysis of various state-of-the-art semantic dependency parsing systems. The analysis will consist of a close examination and comparison of these systems, which will provide insight to, and serve as an empirical foundation and basis for our own research. The area that we found most promising as a focal point for our own research proved to be semantic frame classification.

In order to build a state-of-the-art semantic frame classifier, we will, firstly, examine a set of machine learning algorithms, and secondly, perform an in-depth feature selection based on available data sets for semantic frame classification. We hope to demonstrate that the results of our classifier can be used as a basis for improving the accuracy of current semantic dependency parsers by improving their frame classification accuracy.

1.1 Overview

Chapter 2 provides background for our thesis. In this chapter we briefly outline dependency grammar, and examine various approaches to dependency based parsing. We differentiate between grammar- and data-driven dependency parsing, and formally describe both of these approaches. We divide data-driven dependency parsing in two main classes: transition-based and graph-based models. We examine these two models and give examples of their implementation. Finally, we define semantic dependency parsing, and demonstrate how this approach differs from syntactic dependency parsing.

Chapter 3 examines a few selected state-of-the-art semantic dependency parsing systems. We have chosen to focus on a set of data-driven parsers that participated in Task 18 at SemEval 2015 on *Broad-Coverage Semantic Dependency Parsing* (SemEval-2015) (Oepen et al., 2015). We present the parsing systems that where part of this task, and provide the reader with their technical details. We will also examine the target representations used for training and testing that were part of SemEval-2015. Here we also examine Task 8 at SemEval 2014 (see Oepen et al. (2014)), which was the predecessor to SemEval-2015.

Chapter 4 builds on the previous chapter by examining the results submitted by a subset of the parsing systems participating in SemEval-2015. We perform an in-depth contrastive error analysis of the parsing systems in order to gain insights into common errors shared by these systems. We examine four factors of semantic dependency parsing: (1) length factors, (2) graph factors, (3) linguistic factors

and (4) frames accuracy. The insights from this analysis will be the empirical foundation for our experiments.

Chapter 5 presents the experimental setup we use as basis for our semantic frame classification task. We present the data sets used for training, development and testing. We then present the type of features that we will focus on in our experiments, and the reasoning behind the feature selection. Lastly we present a description of the machine learning algorithms we employ in our experiments.

Chapter 6 presents the results of our experiments. We start the chapter by establishing a baseline score for semantic frame classification that we use as the comparative basis for evaluating the effects of our feature selection. We then present the results of each set of features and their impact on the overall accuracy of our classifier. In our final experiments we show that we can achieve state-of-the-art accuracy using an experimental approach where a large feature space is explored. We show that the results of our semantic frame classifier can be used in conjunction with the highest scoring semantic dependency parsers participating in SemEval-2015. We demonstrate that we can increase these system's semantic frame classification accuracy. We thus conclude that our results have proved fruitful in pushing the state-of-art in semantic dependency parsing one small step forward.

Chapter 7 functions as concluding remarks to our thesis. We summarize our main contributions, and discuss possible future work.

Chapter 2

Background

This chapter provides an introduction to dependency grammar and reviews the state-of-the-art in dependency-based parsing. Firstly, we will look at the topic of dependency grammar in order to provide the reader with some background, and additionally as a means to formally define a set of properties that will be used as basis for our thesis. Secondly, we will examine the two main approaches to dependency-based parsing: the *grammar-driven* and the *data-driven* approach. These two approaches are not mutually exclusive. Lastly, we will shed some light on a few models that are based on a combination of both.

The focus of this chapter will be on the data-driven approaches to dependency parsing, as most of the recent research on dependency-based parsing use this approach. In addition, the contrastive errors analysis that we present in chapter 4 use data-driven parsers and their results as its empirical foundation. The experiments that we present in chapter 5 similarly rely on data-driven models for classifying semantic frames.

We will focus on two main approaches to data-driven parsing: *transition-based* and *graph-based*. The transition-based approach, also commonly referred to as *shift-reduce dependency parsing*, is based on *finite state machines* for mapping a sentence to a dependency graph. The learning aspect of this approach is to parametrize over local transition states: a model for predicting the next state given previous states. The parsing use this model to construct the most optimal sequence of transitions for a given sentence in order to reach a dependency structure. The graph-based approach to parsing, also known as *maximum spanning tree* parsing, create a space of candidate dependency graphs for a given sentence. This approach to learning is global, and the parsing aspect is to search through a set of dependency graphs and carry out a weighted selection in order to reach the most probable dependency structure.

We differentiate between two classes of dependency-based parsing: *syntactic dependency parsing* and *semantic dependency parsing*. As a superficial starting

point we can state that the dependency relations in the former are represented predominantly as *tree* data structures, while the latter are represented as *graph* data structures. In section 2.3, we examine the hypothesis claiming that tree-based data structures are suited for the analysis of grammatical structure, but often lack the expressiveness needed to capture semantics due to the limitations imposed by its structure. We will argue that semantic dependency parsing deserves further investigation in light of these observations.

In order to simplify our discussion, we will use the term graph when discussing dependency grammar in section 2.1. This is based on two assumptions:

1. A tree can be defined as an acyclic directed graph; all possible trees are a subset of all possible graphs.

2. The difference between tree-based and graph-based dependency structures are only of interest to our discussion when dealing with parsing techniques and algorithms, and not when discussing dependency grammar.

We do not attempt at providing a comprehensive review of dependency grammar, nor an in-depth formal description of the various approaches and algorithms to dependency-based parsing. Our main objective in this chapter is, however, to present the reader with the necessary background and context for our research and analysis in subsequent chapters. Firstly, we will provide a brief review of dependency grammar.

2.1 Dependency Grammar

The early roots of dependency grammar can possibly be traced back to Pāṇini's grammar of Sanskrit written in approximately 350-250 BC (Kruijff, 2002). However, the modern study of dependency grammar is first presented in the works of Tesnière (2015 [1959]). In his seminal work, *Elements of Structural Syntax*, Tesnière presents a theory of syntax by focusing on what he calls a *connection* and a *dependency*:

> The sentence is an *organized whole*; its constituent parts are the *words*. Every word that functions as part of a sentence is no longer isolated as in the dictionary: the mind perceives *connections* between the word and its neighbors; the totality of these connections forms the scaffolding of the sentence. The structural connections establish relations of *dependency* among the words. Each such connection in principle links a *superior* term and an *inferior* term. The superior term receives the name *governor (régissant)*; the inferior term receives the name *dependent (subordonné)* (Tesnière, 2015 [1959]).

It is these *connections*, according to Tesnière, that make a sentence meaningful: "[W]ithout them the sentence would not be intelligible" (Tesnière, 2015 [1959]). The *connections* are used to create a hierarchy between the words in a sentence, where one word is dependent on another, hence the term *dependency*. This is a different approach than for instance *constituency grammar*, where the relationships between lexical units are formed under grammatical constituents. In constituency grammar there are also no hierarchical relationship between the lexical units, instead the hierarchy is formed as grammatical groups such as the *sentence*, *noun-phrase*, *verb-phrase*, etc. In dependency grammar the lexical units are always atomic, and the dependency relations are all bi-lexical.

From the works of Tesnière, the field of dependency grammar has grown into a wide range of traditions that have explored the notion of dependency from a variety of different perspectives. Among these are the Prague School's Functional Generative Description, Meaning-Text Theory, and Hudson's Word Grammar (Sgall, Hajičovà, & Panevovà, 1986; Mel'čuk, 1988; Hudson, 1990). We will not provide a detailed exposition on the differences and similarities between the various approaches to dependency grammar, but rather focus on the aspects that are informative as a precursor to our section on dependency parsing. What follows is a concise and formal definition of dependency grammar, and a set of criteria that can be used for determining dependencies in a sentence. We also introduce important terminology, that will be used throughout our thesis, relating to dependency grammar.

2.1.1 Defining Dependencies

A dependency can be described as a binary asymmetrical relation between the lexical units of a sentence, i.e. as arrows pointing from one lexical unit to another. Formally, we describe the dependencies in a sentence $\vec{w} = w_1...w_n$ as a directed graph on the set of positions $\vec{w}$ that contain an edge $i \rightarrow j$ if and only if w_j depends on w_i (Kuhlmann, 2010). Directed edges between the lexical units are used in order to represent a dependency going from one lexical unit to another.

The common term used in the research literature for the lexical unit that stands at the beginning of the arrow is *head*. For the lexical unit that is pointed to by the arrow-head the term *dependent* is the most common. An exception to the description thus far is the lexical unit that acts as the entry-point of the graph, usually the main verb of the sentence. In order for this lexical unit to have a head, a common strategy is to add an artificial unit to the sentence, often named *root*. The edges in the graph can have *labels* added, which signify the type of relation that exists between heads and dependents.

If we examine the dependency graph in figure 2.1, we can visualize the description above: the verb 'brought' that is the entry-point of the graph, the ar-

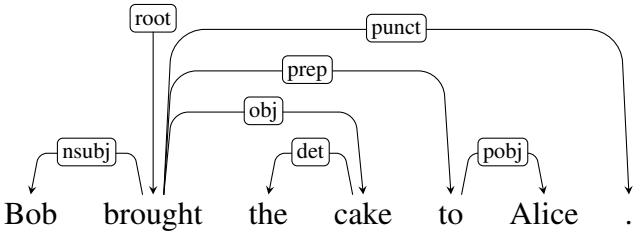

Figure 2.1: Example dependency graph with labeled edges.

tificially added unit *root* that acts as the head of the verb, and several labeled dependencies between the heads and dependents in the sentence. If we examine the dependency between the verb 'brought' and the noun 'Bob': we see that the head of this dependency is the verb 'brought', the dependent is the noun 'Bob', and the edge that connects them has the label 'subj'. The label in this example is used to encode syntactic information; 'Bob' acts as the nominal subject of the verb 'brought'. Now that we have briefly defined the dependency graph, we turn our attention to a set of criteria that has been proposed for determining heads and dependents in a sentence.

2.1.2 Criteria for Dependencies

The criteria for establishing the dependencies, i.e. determining which lexical units should be head and dependents in a sentence, are of central concern to dependency grammar. Nivre (2005a) proposes a set of criteria, with reference to Zwicky and Hudson, for establishing dependencies and determining the head H and dependent D in a construct C (Zwicky, 1985; Hudson, 1990; Nivre, 2005a):

1. (H) determines the syntactic category of (C) and can often replace (C).
2. (H) determines the semantic category of (C), whereas (D) gives semantic specification.
3. (H) is mandatory, whereas (D) can be optional.
4. (H) selects the category of (D) and whether it is mandatory or optional.
5. The form of (D), whether it is agreement or government, depends on (H).
6. The position of (D) is specified in relation to (H).

Different traditions of dependency grammar diverge in their interpretation and use of a specific set of criteria for identifying dependencies. The list above encompasses a set of syntactic and semantic criteria for establishing dependencies, and there have been attempts at providing a single coherent notion of dependency that include all of the criteria above.

Hudson has proposed the usage of the concept of a prototype structure that satisfies all or most of the criteria above, and then using special cases for dependencies that only satisfy one or few criteria (Hudson, 1990). In contrast, Mel'čuk proposes a set of three dependency types: *morphological*, *syntactic*, and *semantic* (Mel'čuk, 1988). Lastly, Nikula suggests two categories of constructions, namely *endocentric* and *exocentric*, for determining dependencies (Nikula, 1986). *Valency* is another term that is used as criteria for determining dependencies. We will now define the terminology thus far, and add additional terms that are used in the subsequent chapters relating to dependency grammar.

Endocentric Construction This is a term used for constructions where the dependent is optional and not selected by its head, and where the head can replace the whole without affecting the syntactic structure of the sentence. In terms of the categories above, they are all endocentric with the exception of number 4. (Kübler, McDonald, & Nivre, 2009). The term *head-modifier* is often used in the research literature to describe a construct where the head modifies the dependent either syntactically or semantically. Head-modifier constructs usually fall within the definition of an endocentric construction (Nivre, 2005a).

Exocentric Construction These are constructions that fail on the first criteria, where the head can substitute the whole construct, but may satisfy the others. The term *head-complement* is often used in recent syntactic theories to describe an endocentric construct (Nivre, 2005a).

Valency The term *valency* is used to determine the distinction between complements and modifiers. In most theoretical frameworks the term valency is used in relation to the semantic predicate-argument structure that is associated with verbs, nouns and adjectives (for the most part). It is used as a way to describe a construct where the lexeme impose some type of requirement on its dependent that determines how to interpret it as a semantic predicate (Nivre, 2005a).

Projectivity This is a technical term that sets a boundary on the type of dependencies that are permissible in a graph. A dependency graph is projective if and only if for all its edges $w_i \rightarrow w_j$ in a sentence $\vec{w} = w_1 ... w_n$, they adhere to the restriction that if $w_i \rightarrow w_k$ then $i < k < j$ when $i < j$, or $j < k < i$ when $j < i$

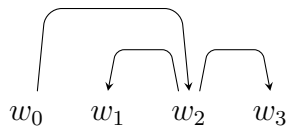

Figure 2.2: A projective dependency graph.

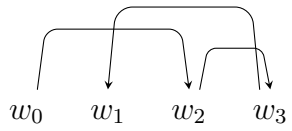

Figure 2.3: A non-projective dependency graph.

(Kübler et al., 2009). We can see the difference by examining the projective graph in figure 2.2, and the non-projective graph in figure 2.3.

Single-head constraint This term is used to define a constraint where the dependents in a sentence are prohibited from having more than one head. This limitation reduces the flexibility of a dependency graph, and as we shall see in section 2.3 on syntactic and semantic dependency parsing, this constraint reduces the possibilities of capturing certain semantic information.

There are a range of different traditions and theoretical frameworks of dependency grammar. What we have presented here is a short introduction to some aspects of these theories as a foundation for section 2.2 on dependency-based parsing, and section 2.3 where we examine the difference between syntactic and semantic parsing. We do not dive into the finer details of dependency grammar as these are not seen as relevant for our thesis. As Nivre points out, the theories of dependency grammar are only indirectly linked to the techniques used in dependency parsing, and the connection between dependency grammar and dependency-based parsing is largely indirect. Nivre states that one should think of dependency-based parsing as parsing with dependency *representations* rather than with a dependency grammar (Nivre, 2005a). In addition, the work presented in our thesis is based on practical considerations such as the effectiveness and accuracy of different parsing techniques. The grammatical aspects are of interest as they can add to our discussion regarding these aspects of dependency-based parsing, but they are not of central concern to our thesis.

Now that we have given an outline of dependency grammar, we turn our at-

tention to dependency-based parsing. In doing so, we follow Carroll (2000) in distinguishing between two main approaches to dependency parsing, a *grammar-driven* approach and a *data-driven* approach. both approaches aim to produce a dependency structure for a given sentence by algorithm (an example of which is seen in table 2.1). The methods applied to reach this goal are, however, different.

2.2 Dependency Parsing

The early approaches to dependency parsing were based on formal grammars in order to automatically assign a dependency structure for a given sentence. However, as Nivre points out, even though some dependency parsers are intimately tied with a particular theory of dependency grammar, it is more often the case that a parser is based on a *representation* rather than a formal theory. Constituency based parsing, in contrast, is often more tied to a particular theoretical approach (Nivre, 2005a).

In the more recent literature on dependency parsing there has been a shift towards data-driven approaches. This is due to the fact that these approaches have consistently shown progress in both accuracy, speed and robustness. The data-driven approaches are based on statistical modeling or machine learning algorithms for inducing probabilistic or predictive models. We start this section by giving a short review of grammar-driven dependency parsing, before we move on to examining the data-driven approach.

2.2.1 Grammar-Driven Approaches

Grammar-driven dependency parsing relies on explicitly defined grammars for producing a dependency graph. Given a sentence, a strategy is deployed in order to find a dependency structure that belongs to the language defined by a specific grammar. This grammar can be made manually, by means of statistical modelling or machine learning, or a fusion of both.

The earliest works on dependency parsing were closely related to context-free grammars (Kübler et al., 2009). These methods use production rules in a context-free grammar in order to produce dependencies. Standard chart parsing methods are used for implementation, examples of which are the Cocke-Kasami-Younger (CKY) (Younger, 1967) and Earley's algorithm (Earley, 1970). The rules themselves can take the form of production or constraint rules; see Kübler et al. (2009) for details.

Gaifman proposes a set of three rules for a *dependency system*. The rules are similar to context-free grammars in that they map a sentence $\vec{w} = w_1...w_n$ to a sequence of categories $X_1, ..., X_n$, and they add a relation of dependency d

between two lexical units $w_i \rightarrow w_j$ as long as a set of conditions or constraints are upheld. The rules that Gaifman propose lead to a dependency structure that is a projective directed tree which upholds the single-head constraint (Gaifman, 1965). These approaches produce unlabeled dependency structures.

Nivre points out that the results from these early approaches, and the attempts of Gaifman to show that dependency grammar is only a restricted variant of context-free grammars, led to a period of approximately twenty-five years with a relative lack of interest in dependency parsing among researchers working in the field of NLP (Nivre, 2005a).

Another common approach to grammar-driven dependency parsing is based on what is commonly referred to as *eliminative* parsing. This approach, as opposed to the systems based on context-free grammars, produce dependency structures by continuously eliminating dependencies that violate a set of constraints. The elimination process is repeated until there are no violations. As Nivre notes, the eliminative approach is a constraint satisfaction problem, where all dependency structures that are not in violation with the constraints, would be considered. This approach poses two problems. The first problem is that the result might not be a dependency structure at all, i.e. all suggestions break some constraint. The second problem arises when more than one dependency structure remains (Nivre, 2005a). The latter problem can be solved using a disambiguation step. We will take a closer look at disambiguation in section 2.2.2 below.

Recent research on dependency parsing suggests a move away from grammar-driven approaches, as data-driven parsing, increasingly, show improvements in both accuracy, speed and robustness. In light of this observation, and secondly, because both the analysis in chapter 4, and our own experiments in chapter 5, are based on data-driven approaches, we will not examine the grammar-driven approach in any great detail. Instead, we turn our attention to the data-driven models of dependency parsing.

2.2.2 Data-Driven Approaches

Early attempts at data-driven dependency parsing used a grammar-driven part to produce multiple dependency graphs, and added a data-driven model trained on corpora to select the most probable structure from a set of possibilities. Eisner presented one of the first successful approaches using this methodology. This approach use three models for probabilistic parsing, trained on manually annotated data, where both part-of-speech tags and unlabeled dependency structures are assigned to a sentence (Eisner, 1996). The algorithm used, similar to the CKY method, is a bottom-up method that predicts the most probable parse from the bottom up. It ensures that there are no cycles in the graph, and adheres to the single-head constraint defined in section 2.1.1. Even though this parser is data-driven,

in the sense that no hand-written grammar is required, the bottom-up strategy is based on a learned grammar that in combination with a generative probabilistic model attempts to predict the most likely parse for a given sentence.

As Nivre points out, the work of Eisner has been influential in two ways. It proved that statistical modeling and machine learning could be used for dependency parsing with an accuracy comparable to the best performing constituency-based parsers of the time. Secondly, it revealed that efficient parsing techniques exploiting the special properties of dependency structure could be developed (Nivre, 2005a). More recent approaches have moved towards data-driven models where the dependency graph is induced without any explicitly defined grammar. We will now take a closer look at the two main classes of data-driven dependency parsing.

Transition-Based Parsing

Purely deterministic discriminative data-driven models with no need for a grammar were first proposed by Kudo and Matsumoto (2000) and Yamada and Matsumoto (2003). These models use *Support Vector Machines* (SVMs) for learning, and rely on the machine learning algorithms' ability to cope with large scale feature spaces. The parsers presented by Kudo and Matsumoto and Yamada and Matsumoto construct dependency trees in a left-to-right fashion by way of three transitions: *Shift*, *Right*, and *Left*. Yamada and Matsumoto use three binary classifiers in order to solve these transitions as a multi-class classification problem. One model is made for handling each possible action given a state: *Left* vs. *Right*, *Left* vs. *Shift* and *Right* vs. *Shift* (Yamada & Matsumoto, 2003). This method managed to produce substantially higher accuracy than the models proposed by Eisner (1996).

The models developed by Kudo and Matsumoto and Yamada and Matsumoto have been further developed in many directions under the umbrella of transition-based parsing. A *transition-based* parser consists of a set of *configurations* (or *states*) that include a set of *transitions* for producing a dependency structure.

We follow Kübler et al. (2009) in describing transition-based parsing as a *configuration* of triples, consisting of a stack, an input buffer, and a set of dependency arcs. Given a set R of dependency types and a vocabulary V, a *configuration* for sentence $S = w_0 w_1, ..., w_n$ is a triple $c = (\alpha, \beta, A)$, where:

1. α is a stack of words $w_i \in V_S$,
2. β is a buffer of words $w_i \in V_S$,
3. A is a set of dependency arcs $(w_i, r, w_j) \in V_S \times R \times V_S$.

The *configuration* represents a partial analysis. The words on the stack α are partially processed words from the input, and the words in the buffer β are

the remaining words from the input. For any input sentence there is an *initial* state, and a *termination* state. The initial state starts with the artificially added unit *root* (w_0) on the stack α, the input sentence S in the buffer, and an empty set of dependency arcs in the last place of the triple: $([w_0]_\alpha, [w_1, w_2, ..., w_n]_\beta, \emptyset)$. The process ends in the *termination* state: $(\alpha, []_\beta, A)$. There are three types of transitions from the initial to the termination state:

1. $Left - Arc_r$ $\quad (\alpha|w_i, w_j|\beta, A) \Rightarrow (\alpha, w_j|\beta, A \cup \{(w_j, r, w_i)\}$
2. $Right - Arc_r$ $\quad (\alpha|w_i, w_j|\beta, A) \Rightarrow (\alpha, w_i|\beta, A \cup \{(w_i, r, w_j)\}$
3. $Shift$ $\quad (\alpha, w_i|\beta, A) \Rightarrow (\alpha|w_i, \beta, A)$

Each of these transitions can be described informally as:

1. Left-Arc: add a dependency arc (w_i, r, w_j) to the set A, where w_i is a lexical unit on top of stack α and w_j is the first lexical unit in buffer β. Then pop the top lexical unit from stack α.
2. Right-Arc :add a dependency arc (w_j, r, w_i) to the set A, where w_i is a lexical unit on top of stack α and w_j is the first lexical unit in buffer β. Then pop the top lexical unit from stack α. Then replace w_j by w_i at the head of buffer.
3. Shift: remove the first lexical unit w_i in the buffer β and push it on top of the stack α.

The transitions are performed according to a set of permissible transitions in a sequence T. This includes a sequence of configurations $C_{0,m} = (c_0, c_1, ..., c_m)$ for sentence S where:

1. c_0 is the initial configuration $c_0(S)$ for S,
2. c_m is a terminal configuration,
3. for every i such that $1 \leq i \leq m$, there is a transition $t \in T$ such that $c_i = t(c_{i-1})$

An example of a transition-based dependency parser is the openly available MaltParser (Nivre et al., 2007). This parser is an implementation of the *inductive dependency parsing* techniques developed by Nivre (2005b). Given a *treebank* that follows the specific dependency format of the MaltParser, it can induce a parser for the language of that treebank. MaltParser itself includes two basic parsing algorithms, but it also provides an interface so that a variety of shift-reduce based algorithms can be used for parsing. For more details on the algorithms and the interface see Nivre et al. (2007).

Graph-Based Parsing

Graph-based parsing employ already established and extensively studied graph processing algorithms to generate a dependency graph for a sentence. In contrast to the transition-based approach where the learning is locally trained, graph-based approaches parametrize models globally on substructures of a dependency structure. The main aspect of the parsing is then to give a *score* to each substructure, and return the structure with the highest score.

The scoring function is at the core of the graph-based approach. Scores can be calculated using linear classifiers, or use conditional or joint probabilities. The *arc-factored model* is the simplest of the graph-based dependency parsing approaches. It is often referred to as a first-order model, due to its scoring function where possible graphs are evaluated one edge at a time. There are also second- and third-order models where the scoring function decompose the graph in larger fragments, and calculate the value of each graph in more complex manners. The scoring also has a set of restrictions that prohibits the parser from producing invalid outputs (Kübler et al., 2009).

The Mate parser (Bohnet, 2010) is an openly available second-order graph-based dependency parser. The parser employs a second order maximum spanning tree algorithm, a modification of the algorithm found in Carreras (2007), and combine this with a passive-aggressive perceptron algorithm. The Mate parser, as we will see in chapter 3, has proven to consistently show exceptionally high accuracy for semantic dependency parsing of the English language.

There are a number of parsers, and descriptions of systems, that follow on the transition- or graph-based approaches described above. Different statistical modeling or machine learning algorithms are used for the learning, the same which applies for the parsing actions. In Chapter 3, we will take a closer look at several state-of-the-art semantic dependency parsing systems. Before diving into this discussion, we will give an overview of semantic parsing, and show how it differs from syntactic dependency parsing.

2.3 From Syntactic to Semantic Parsing

The most common representations in dependency parsing in the research community are based on tree data structures. Some scholars argue that such representations are insufficient, in that they restrict the type of dependency structures that are possible, and do not fully capture what can be expressed in natural language. Hudson claims that representations of relative clauses, control relations, and other long-distance dependencies, can only be represented properly by us-

ing more general graphs (Hudson, 1990). Similarly, Sagae and Tsujii argue that tree data structures cannot fully capture linguistic phenomena beyond so-called shallow syntactic structures (Sagae & Tsujii, 2008).

Oepen et al. (2014) argues that tree-oriented parsers are ill-suited for producing meaning representations, as such parsers lack the ability to capture some aspects of semantic analysis. This is particularly evident in cases where a lexical unit is the argument of multiple predicates, or when dealing with semantically vacuous word classes that should be left out of the dependency structure. Oepen et al. (2014) argue for a move towards semantic dependency parsing using graph data structures to enable a more direct analysis of *Who did What to Whom?*

Approaches to semantic dependency parsing are very similar to those of syntactic dependency parsing, and many of the successful ones are based on a variation of the models described in this chapter. An early such approach is the modified shift-reduce algorithm proposed by Sagae and Tsujii (2008). They introduce a data-driven approach to dependency parsing where directed acyclic graphs (DAGs) are produced directly from an input string by introducing a modified version of the transition-based model described in section 2.2.2. Two new transitions are introduced to the three transitions that we have already described above: *left-attach* and *right-attach*. These transitions can be described informally as:

1. Left-Attach: add a dependency arc (w_i, r, w_j) to the set A between the top two items on the stack α, making the top item w_i the head and the item below it w_j the dependent, if and only if there is no arc between them already.

2. Right-Attach: add a dependency arc (w_i, r, w_j) to the set A between the top two items on the stack α, making the top item w_i the dependent and the item below it w_j the head, if and only if there is no arc between them already. Remove the top item on the stack α and place it back on the buffer β.

This novel approach makes it possible to represent certain semantic aspects of a sentence that is not possible with syntactic dependency parsing, where such restrictions such as the single-head constraint is put in place. As Sagae and Tsujii (2008) explain, there are also other semantic aspects of a sentence that semantic dependency parsing manages to capture, such as anaphoric reference and semantically motivated predicate-argument relations. Until recently, transition-based parsing was mainly restricted to tree based parsing. In Chapter 3 we will examine modifications to transition-based systems that make it possible to handle graphs.

2.4 Conclusions

In this chapter we have briefly outlined the history of dependency grammar and parsing, with particular emphasis on grammar-driven and data-driven dependency

parsing, in addition to an introduction to the different approaches within these two parsing traditions. We have provided definitions for certain technical aspects of dependency parsing that are deemed as central to our thesis.

In the following chapter, we will examine the state-of-the art in semantic dependency parsing. We will focus on a group of parsers that are state-of-the-art in terms of their accuracy. We will also review the annotated data that have been used for training, development and testing of these parsers.

Chapter 3

Semantic Dependency Parsing with Frames

This chapter presents the state-of-the-art in semantic dependency parsing. The goal of semantic dependency parsing can be superficially defined as a representation of 'Who did what to whom', possibly adding 'when' and 'where' to the equation. This is similar to the aims of Semantic Role Labeling (SRL), which according to Jurafsky and Martin (2009) is a shallow semantic representation of *semantic roles*. They define semantic roles as the abstract role that the argument of a predicate can take in an event. Semantic dependency parsing usually has a broader scope in its representation. In addition to the goals of SRL, semantic dependency parsing attempts to identify various semantic phenomena, such as negation, topicalization, relative clauses, and other scopal dependencies that are not part of the scope of SRL.

More specifically, this chapter will discuss the definition of *Broad Coverage Semantic Dependency Parsing* made by Oepen et al. (2015), which reads as follows:

> ... [T]he problem of recovering sentence-internal predicate-argument relationships for *all content words*, i.e. the semantic structure constituting the relational core of sentence meaning.

The emphasis on predicate-argument dependencies for all content words is the reason for focusing on dependency parsing techniques that can output a graph structure instead of a tree. In Chapter 2 we ended with a note on this aspect of dependency parsing, and in this chapter we will clarify and expand further upon where we left off.

As previously discussed, the target representations used in the research on syntactic dependency parsing, and the results that such parsers are able to produce,

have been largely limited to tree data structures. A tree can be defined as an acyclic directed graph, i.e. every node is reachable from a root node by exactly one directed path. This structure impose certain restrictions, such as a unique root, connectedness, and lack of reentries (the so-called single-head constraint), on the type of dependencies that can be represented.

As a result, the restrictions that trees impose limit certain aspects of semantic analysis, such as the analysis of dependents with more than one head, and the possibility to leave certain semantically void lexemes outside of the dependency structure.

In order to mitigate the restrictions imposed by tree-based parsing, efforts have been made to develop graph-structured target representations. Data-driven parsing techniques have been developed that can be trained on representations where the annotations of a sentence establish a dependency graph. These parsing techniques are therefore capable of producing dependency structures that are better apt at capturing sentence semantics than parsing techniques that output a tree structure.

We will argue our case by presenting the results of SemEval-2014 Task 8 and SemEval-2015 Task 18 on Broad Coverage Semantic Dependency Parsing (referred to as SemEval-2014 and SemEval-2015 from now on respectively) (Oepen et al., 2014, 2015). The two shared tasks have provided a large annotated corpora in 4 different annotation schemes for training and testing. These are annotations on the Wall Street Journal (WSJ) corpus of the Penn Treebank (PTB) for SemEval-2014, and the added Brown corpus of the same Treebank for SemEval-2015 (Marcus, Santorino, & Marcinkiewicz, 1993). Several researchers submitted their results to the two shared tasks, with many achieving state-of-the-art accuracy on their results.

We will first examine the target representations made available for SemEval-2014 and 2015, and then move on to a presentation of the technical aspects of the various submissions. We restrict the technical presentation to the SemEval-2015 task, which we will also do when examining higher level statistics on the data sets. Since the SemEval-2015 task is an extension of the previous year, we find it more useful to focus our attention on the data sets and submissions of that year. The results also saw an increase in overall performance in the SemEval-2015 shared task than the previous year. In Chapter 4 we follow the same suit and focus solely on the SemEval-2015 results in our contrastive error analysis of the results of a selected group of teams and their submissions.

Though SemEval-2015 include annotated corpora for Czech and Chinese (Mandarin) languages, in addition to English, we have decided to limit our research and analysis to the English language, and will therefore not include these results in our thesis. It is, however, worth mentioning that these target representations exist, and that several submissions to the SemEval-2015 task also submitted results from parsers trained on the target representations in these two languages.

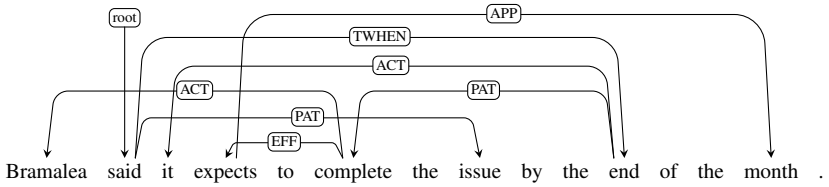

Figure 3.1: PCEDT target representation.

3.1 Target Representations

Target representations are an integral part of data-driven parsing. They are the foundation on which data-driven parsers are trained on in order to predict the most plausible dependency structure for a given sentence. The four distinct representations we will examine use different annotation schemes. The first three representations we will examine are called DM, PAS and PCEDT, which were used for the SemEval-2014 shared tasks. For the SemEval-2015 task, the PCEDT target representation was replaced by PSD, and so-called *Frames* where added to the DM and PSD representations. In Tables 3.3, 3.2, 3.1 and 3.4 we have visually represented the annotations of DM, PAS, PCEDT and PSD on the sentence:

> Bramalea said it expects to complete the issue by the end of the month.

This sentence has been chosen in order to highlight certain aspects of a semantic dependency graph: some lexical units are left unattached, we have a few examples of lexical units with more than one head (breaking the so-called single-head constraint that a tree would impose), and dependencies that cross, making the graphs non-projective. It is worth noting that a tree can be non-projective, and that this is not a special case for graph representations, but is often desired in order to fully represent longer predicate-argument dependencies that create crossing dependencies across a sentence.

The data-sets that we will examine are all represented in the SDP format[1]. We will now examine how these target representations have been constructed, and present some higher-level statistics on their content.

[1]See `http://sdp.delph-in.net/2014/data.html` and `http://sdp.delph-in.net/2015/data.html` for the technical details on the data format of SemEval-2014 and 2015 respectively.

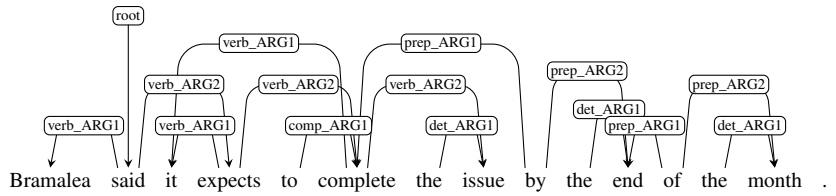

Figure 3.2: PAS target representation.

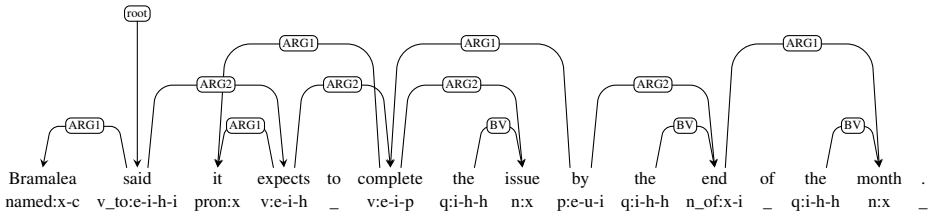

Figure 3.3: DM target representation.

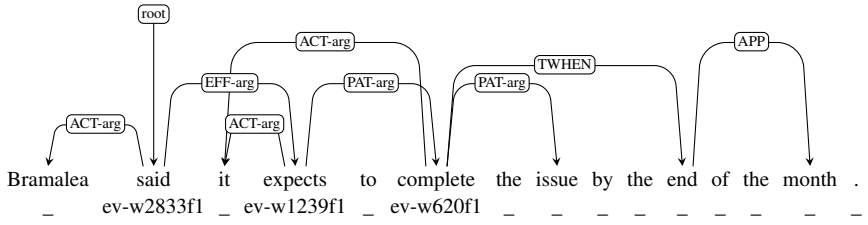

Figure 3.4: PSD target representation.

PCEDT: Prague Tectogrammatical Bi-Lexical Dependencies This target representation is based on the Prague Czech-English Dependency Treebank (Hajič et al., 2012)[2]. It is a dependency treebank over the WSJ from the PTB. The original English texts have been annotated along with annotated Czech translations. Similar to other treebanks from the PTB, these texts have been annotated with two layers of syntactic information: *analytical* (a-layer) and *tectogrammatical* (t-layer) (Oepen et al., 2014). The a-layer represents the so-called surface syntax, where the labels in the dependencies represent the syntactic information of the sentence. The t-layer is a layer representing syntax and semantic dependencies are represented, and is based on the framework of the Functional Generative Description (Sgall et al., 1986). A conversion has been used in order to reach the SDP data format from this t-layer; see Miyao, Oepen, and Zeman (2014) for details on the conversion from the t-layer of the PCEDT representation to the SDP representation.

PAS: Enju Predicate–Argument Structures The Enju representation is based on Head-driven phrase structure grammar (HPSG), and is derived from the Enju HPSG treebank. This treebank is made by way of conversions from the phrase structure and predicate-argument representation of the PTB (Oepen et al., 2014). The PAS representation is extracted from the predicate-argument structures of the HPSDG Treebank. This predicate-argument structure represent bi-lexical semantic dependencies. As the PCEDT format, we refer the reader to Miyao et al. (2014) for the technical details on the conversion to the SDP data format.

DM: DELPH-IN MRS-Derived Bi-Lexical Dependencies The semantic dependency graphs of the DM format are derived from the output of the ERG parser. This parser adds syntactic and semantic analysis by using the LinGO English Resource Grammar (LERG). LERG is, in a similar fashion to PAS, based on HPSG. It adds to the standard framework of HPSG by using Minimal Recursion Semantics for specifying semantic attributes, but does so without implementing the binding theory of HPSG (Flickinger, 2000). The DM representations are derived through a two-step 'lossy' conversion. The first step in this conversion is to convert the MRSs to variable-free *Elementary Dependency Structures*. The second step is to transform the results of the previous step to the strictly bi-lexical SDP data format. During this last step some information may be lost (Miyao et al., 2014).

[2]See `http://ufal.mff.cuni.cz/pcedt2.0/`

PSD: Prague Semantic Dependencies The PCEDT target representation is used as basis for arriving at the PSD[3] target representation. This is made using a conversion. The PCEDT representation consists of dependency structures that are always rooted trees. This is due to technical aspects of the conversion from the t-layers mentioned above, to the PCEDT data format. For the SemEval-2015 task, a conversion of the PCEDT data's t-layer was performed in order to reach true bi-lexical dependencies.

3.1.1 Semantic frames

The SemEval-2015 introduced frames, also referred to as sense distinctions, to the DM and PSD target representations. These are added as an extra layer to the sentence where multiple classes are used in order to add additional information on the content words of a sentence. If we examine the semantic dependency graphs in Figure 3.3 and 3.4, we see these classes below the sentence.

According to Oepen et al. (2016), DM frames encode more general 'linking patterns', which are mappings from syntactic to semantic arguments, whereas PSD represents actual sense identifiers and show different values for distinct lexemes. They note further that PSD only annotates senses on verbal predicates, while DM provide frame identifiers for all semantically contentful nodes. Additionally, Oepen et al. (2015) note that the DM frames are limited to argument structure distinctions, e.g. causative vs. inchoative contrasts or differences in the arity or coarse semantic typing of argument frames. They further note that the PSD frames draw on much richer sense inventory, based on the EngValLex database (Cinková, 2006).

As we will see in Chapter 4, our analysis of the results of SemEval-2015 will lead us to further explore semantic frames and build our own *frame classifier*. The details of this classifier and its results will be presented in Chapter 5.

3.2 Data sets

The data-sets for the SemEval-2014 and 2015 vary slightly. In the SemEval-2014, the three annotations are over the same set of texts: Sections 00-21 of the WSJ corpus. A set of sentences where excluded from the data sets where (a) no gold-standard analysis existed; (b) it was not possible to align the tokens of each sentence on-to-one for all three representations; (c) there where cycles in at least on of the graphs of a sentence. After this cleanup, the SemEval-2014 data set counts

[3]See `http://tinyurl.com/h8dfkcz` for more technical details on the PSD target representation and conversion from PCEDT.

	In-domain			Out-of-domain		
	DM	**PAS**	**PSD**	**DM**	**PSD**	**PSD**
# labels	59	42	91	47	41	74
% singletons	22.97	4.38	35.76	25.40	5.84	39.11
edge density	0.96	1.02	1.01	0.95	1.02	0.99
$\%_g$ trees	2.30	1.22	42.19	9.68	2.38	51.43
$\%_g$ noncrossing	69.03	59.57	64.58	74.58	65.28	74.26
$\%_g$ projective	2.91	1.64	41.92	8.82	3.46	54.35
$\%_g$ fragmented	6.55	0.23	0.69	4.71	0.65	1.73
$\%_n$ reentrancies	27.44	29.36	11.42	26.14	29.36	11.46
$\%_g$ topless	0.31	0.02	–	1.41	–	–
% top nodes	0.996	0.999	1.127	0.985	1.000	1.264
$\%_n$ non-top roots	44.91	55.98	4.35	39.89	50.93	5.27
average treewidth	1.30	1.72	1.61	1.31	1.69	1.50
maximum treewidth	3	3	7	3	3	5
# frames	297	–	5426	172	–	1208
$\%_n$ frames	13.52	–	16.77	15.79	–	19.50

Table 3.1: High-level statistics on the SemEval-2015 data sets

34,004 sentences (or a total of 745,543 tokens) for the training split (Sections 00-20), and 1,348 sentences (or a total of 29,808 tokens) for the test split (Section 21) (Oepen et al., 2014). The SemEval-2015 shared tasks used the same data set, but added a balanced corpus from the Brown Corpus. Also, the DM graphs where extracted from a later and improved release of the DeepBank (version 1.1). The exclusion of sentences from the data sets where a bit lower for SemEval-2015, and the training set counts 35,657 sentences (or a total of 802,717 tokens; roughly 8% more than for SemEval-2014). For the test set 1,410 sentences (or a total of 31,948 tokens) from the WSJ Section 21 was reserved for in-domain testing, and 1,849 sentences (or a total of 31,583 tokens) from the Brown Corpus was reserved for out-of-domain testing (Oepen et al., 2015).

3.2.1 Quantitative Analysis of Data Sets

In Figure 3.1 we present some high-level statistics on the SD 2015 data set. The data are reproduced from Oepen et al. (2015). The PSD representation is the most fine-grained in terms of the linguistic variation of dependency labels with 91

unique labels. DM and PAS are in this respect more coarse-grained and similar, also sharing a more similar naming and type convention.

When examining the percentage of trees, projective vs. non-projective graphs, and reentrancies, we see that PSD is the most 'tree-oriented' of the three target representations. As previously mentioned, the PSD target representation is based on the PCEDT representation, which only consists of dependency structures that are rooted trees.

As we can observe in Figure 3.1, the number of frames in PSD is approximately 18 times that of the DM data set.

3.3 Submissions and Teams

In this section we will examine a set of state-of-the art semantic dependency parsers. As previously stated, we will examine the submissions from the SemEval-2015 exclusively, disregarding the results from SemEval-2014. This choice is based on the observation that many of the same teams submitted for both tasks, and furthermore, that the SemEval-2015 shared tasks are an extension of the previous year, and lastly, because the scores of the SemEval-2015 submitted results saw an increase in overall performance.

We can observe the performance of the SemEval-2015 parsers in Table 3.2, which have been reproduced from Oepen et al. (2015). There where 6 teams that submitted results from their parsing systems. Each team could submit two runs per track. In Table 3.2, we present the best run (if more than one run were submitted).

The submissions could be made to different tracks. There was a *closed* track: where systems could only use gold-standard semantic dependencies distributed by the organizers of the SemEval-2015 shared task for training. In addition, there was an *open* track: here the teams could use other resources, such as a syntactic parser, as long as these did not use any methodology where the gold-standard syntactic or semantic analysis of the tasks' test data had been used in any way. For the open track, the organizers made available already parsed syntactic analysis of the training data. There was also a so-called idealized *gold* track where gold-standard syntactic companion files in a variety of formats where provided for training (Oepen et al., 2015).

The evaluation of each parser is based on the accuracy of the dependency graphs that they produce on the test set mentioned above, measured against the gold-standard testing data. The evaluation itself is based on the metrics:

1. Labeled precision, recall, and F_1, referred to as LP, LR and LF.

2. Complete predications: Which for the DM and PAS target representation means measuring all outgoing dependency edges, and for the PSD target

representation to the ones where the label has an '-arg' suffix. Here too precision, recall and F_1 score is used, referred to as PP, PR, and PF.

3. Semantic frames: This is comprised of a complete predication with scores for the frame (or sense) identifier. As other scores, this score is also represented by precision, recall and F_1 score, referred to as FP, FR, and FF.

4. Both complete predications and semantic frame evaluations are limited to those predicates that correspond to verbal parts of speech, as determined by the gold-standard part of speech.

We will now examine the parsing systems of the four best performing teams of SemEval-2015. We turn to the results of three of these systems in Chapter 4, where we will examine, compare and contrast their results in a comprehensive manner.

3.3.1 Peking

The Peking team used two main approaches to solving the problem of semantic dependency parsing. The first approach consists of modifying transition-based models that are designed for handling trees to handle graphs. The second approach converts the training data, consisting of semantic dependency graphs, into tree structures using so-called tree approximation models. After the conversion is done, well-established methods for parsing dependency trees are used in order to train a parser. When predicting, a tree dependency structure is created, which is then converted to a graph dependency structure in order to reach the semantic dependency graph structure of the SemEval-2014 and 2015 data sets. This approach managed to produce high-quality parsing results. Du, Zhang, Sun, and Wan (2014) note that their experiments demonstrated that graph-based models are more effective than transition-based models, and that a parser ensemble can boost parsing accuracy by taking the multiple outputs of several parsers and picking the most plausible results by way of a voting method.

The transition-based models for SemEval-2014 consist of 5 different transition models, which includes a so-called *naive* approach using the shift-reduce method described in Section 2.2.2, but with a pop_k transition added: remove any element from the stack α. They also use a transition based approach described in (Titov, Henderson, Merlo, & Musillo, 2009), which adds to the standard shift-reduce method a pop transition: remove the top element from the stack α, and a $swap$ transition: remove two top elements from the stack α. The transition-based systems where trained twice on the training data, each sentence trained from start to end, but also backwards, from the last to the first element. With the 5 models, the system can thus return 10 parses of a given sentence.

The graph-based models for SemEval-2014 presented by the Peking team use tree approximations. Du et al. (2014) argue that parsing based on graph spanning is a challenging task due to the fact that the graph structures represented by the data sets of SemEval-2014 are still relatively unexplored. In light of this observation, they developed a graph-to-tree transformation which was used to transform the SemEval-2014 data sets to trees. A tree parser is then used to train a model on the transformed data, and finally a tree-to-graph transformation was developed in order to transform the result of the parser back to the graph data structure of the SemEval-2014 data sets. These steps are *lossy*, i.e. information is possibly degradable in the transformation processes, and can be lost, added or modified in the process.

The ensemble method that Du et al. (2014) employ examines the output of 10 transition-based models, and 9 graph-based models, and use a simple voting scheme in order to combine their outputs. This is based on the frequency of the dependencies when combining all models, and given a threshold, if a dependency exceeds it, it is chosen to be part of the result. For the edges that have been chosen, the most frequent label from the 19 models is added. The models are scored by a weighting scheme, so that the graph-based models have a higher score on their edges and labels, since these proved to produce better overall results.

For SemEval-2015, the Peking team further developed this ensemble method, focusing to a larger extent on graph-based models. Du, Zhang, Zhang, Sun, and Wan (2015) developed a *weighted tree approximation model*, where the graph-to-tree transformation is based on weights for the type of transformation that is to take place. When the weights have been calculated, the transformation from a graph to a tree is solved by using maximum spanning tree (MST) algorithms. For SemEval-2015 the transformation is improved by adding additional labels to the trees for a set of specific cases when an edge is lost in the transformation. This allows for a more accurate tree-to-graph reversal. See (Du et al., 2015) for the details on the transformations.

The same type of voting ensemble used in SemEval-2014 is used to combine the outputs of the 10 transition-based models, 9 tree approximation models, and 4 new weighted models added for SemEval-2015. The results revealed an increase in overall accuracy, demonstrating that the improvements to the tree approximation models were fruitful.

3.3.2 Riga

The team behind the Riga system used the system developed by the Peking team for SemEval-2014 as basis for implementing their own semantic dependency parser with a model for predicting semantic frames for SemEval-2015. According to Barzdins, Paikens, and Gosko (2015), the Riga teams approach involves taking

the Peking system, removing some less essential components, and adding a rule-based classifier for both graph parsing and frame labeling. The added system has been dubbed the *C6.0 rule-based classifier*, and used for both graph parsing and frame labeling. See Barzdins, Gosko, Rituma, and Paikens (2014) for the technical details on the classifier.

The Riga team further developed the approach of Peking by modifying the tree approximation model. Instead of the *lossy* method described above, the Riga team developed a fully reversible depth-first transformation. As a baseline they used the Mate parser, see technical details of this parser here: (Bohnet, 2010), which immediately produced results on par with the highest results from SemEval-2014. For SemEval-2015, the Riga team describe three approaches that improved upon this baseline. These three approaches revolve around the *lossless* tree-approximation method, where the information that might be lost in the graph-to-tree transformation is stored in the edges of the dependency tree. The Mate parser is then used to train a parser given these trees with additional edge information. A restructure algorithm is then used to transform the trees back to graphs (Barzdins et al., 2015).

As already mentioned, a rule-based classifier dubbed C6.0 was used as a basis for developing a classifier for predicting frames. This approach involves a refining of the C6.0 classifier from performing an exhaustive search to a greedy search algorithm with a multi-class classifier as a basis. For training, a simple Laplace ratio prediction method is used to count the number of instances that a feature occurs and does not occur with any given frame in the data set. A greedy search is performed over all classes in order to predict frames. See Barzdins et al. (2015) for the specific technical descriptions on the semantic parsing and frame prediction.

3.3.3 Turku

For SemEval-2014 the Turku team developed a semantic dependency parser by combining several classifiers trained with different machine learning algorithms. The *LIBSVM* package: see Chang and Lin (2011) for a description, which is a binary support vector machine classifier, is used for detecting dependencies. For the dependency labels the *SVM-multiclass* package by Joachims (1999) is used to predict the semantic label of the edges predicted in the previous step. The Turku team participated in the open track of the SemEval-2014 task and used a large corpus of syntactic n-grams, and a *word2vec* word similarity model, to improve the classification of labels by using the cosine similarity measures in the word2vec model: see Kanerva, Luotolahti, and Ginter (2014) for specification on the models and training. The last step in this pipeline involves a classification of top nodes, since these are not classified in the first step. A support vector machine classifier is trained in order to predict the top nodes (Kanerva et al., 2014).

For SemEval-2015 the Turku team improved upon the combined classifier ap-

proach from the previous year. Adding a so-called structured support vector machine, the new classifier would get a higher score by taking into account a more global view while training and classifying, instead of the local view of just the head-dependent relationships between lexical units that had been the previous approach. As Kanerva et al. (2014) explain, their approach when dealing with the dependency relations is to predict each predicate independently, i.e. no other predicates or arguments affect the prediction. However, when predicting arguments for one predicate, a global view is kept for the already predicted arguments for this particular predicate.

The Turku team were the only team that submitted results in both the open and gold tracks of SemEval-2015. They have made their parser openly available for others to use as an off-the-shelf parser[4], which in itself is based on the parser of Bohnet and Kuhn (2012). As we can see in Table 3.2, the Turku parser has the overall highest score of the parsing systems. However, this is in part attributed to the fact that the Turku team where the only team participating the gold track, and thus use gold-standard syntactic companion files as part of the training. If we examine the results for the open track, we see that the Turku submission performed substantially lower than the Lisbon team.

3.3.4 Lisbon

The Lisbon team managed to produce state-of-the art results from their parsing system. As Table 3.2 demonstrates, for the in-domain data set it produced the second highest score in the open track, and fourth highest score in the closed track, and for the out-of-domain data set it produced the second highest score in the open track, and third highest score in the closed track. The Turku parser got a higher score in both data sets, but this can be attributed to the fact that those scores were in the gold track.

The Lisbon team's semantic dependency system is named the *TurboSemanticParser*[5], and is available as open source software. This system is the basis for the submissions for both SemEval-2014 and SemEval-2015.

The *TurboSemanticParser* consists of a feature-rich linear model that parametrize globally over first and second order dependencies (arcs, siblings, grandparents and co-parents). As noted by Martins and Almeida (2014), the Lisbon system cast parsing as a structured prediction problem: "Let x be a sentence and $Y(x)$ the set of possible dependency graphs. We assume each candidate graph $y \in Y(x)$ can be represented as a set of substructures (called parts) in an underlying set S (*e.g.,* predicates, arcs, pairs of adjacent arcs)". A score function f will then decom-

[4] `https://github.com/jmnybl/Turku-Dependency-Parser`
[5] `http://labs.priberam.pt/Resources/TurboSemanticParser`

		DM				PAS				PSD			
	LF.av	LF	LP	LR	FF	LF	LP	LR	PF	LF	LP	LR	FF
Turku#	86.81	88.29	89.52	87.09	58.39	95.58	95.94	95.21	87.99	76.57	78.24	74.97	56.85
Lisbon*	86.23	89.44	90.52	88.39	00.20	91.67	92.45	90.90	84.18	77.58	79.88	75.41	00.06
Peking	85.33	89.09	90.93	87.32	63.08	91.26	92.90	89.67	79.08	75.66	78.60	72.93	49.95
Lisbon	85.15	88.21	89.84	86.64	00.15	90.88	91.87	89.92	81.74	76.36	78.62	74.23	00.03
Riga	84.00	87.90	88.57	87.24	58.12	90.75	91.50	90.02	80.03	73.34	75.25	71.52	52.54
Turku*	83.47	86.17	87.80	84.60	54.67	90.62	91.38	89.87	80.60	73.63	76.10	71.32	53.20
Minsk	80.74	84.13	86.28	82.09	54.24	85.24	87.28	83.28	64.66	72.84	74.65	71.13	51.63
In-House*	61.61	92.80	92.85	92.75	83.79	92.03	92.07	91.99	87.24	–	–	–	–

		DM				PAS				PSD			
	LF.av	LF	LP	LR	FF	LF	LP	LR	PF	LF	LP	LR	FF
Turku#	83.50	82.11	84.26	80.07	42.89	92.92	93.52	92.33	83.80	75.47	77.77	73.31	42.37
Lisbon*	82.53	83.77	85.79	81.84	00.35	87.63	88.88	86.41	80.19	76.18	80.12	72.61	02.25
Lisbon	81.15	81.75	84.81	78.90	00.27	86.88	88.52	85.30	78.47	74.82	78.68	71.31	02.09
Peking	80.78	81.84	84.29	79.53	47.49	87.23	89.47	85.10	74.75	73.28	77.36	69.61	34.28
Riga	79.23	80.69	81.69	79.72	41.88	86.63	87.56	85.72	76.26	70.37	73.23	67.71	40.76
Turku*	78.85	79.01	81.54	76.63	39.15	85.95	86.95	84.98	76.38	71.59	74.92	68.55	38.75
Minsk	75.79	77.24	80.24	74.46	42.18	80.44	83.07	77.96	62.00	69.68	72.26	67.27	41.25
In-House*	59.24	89.69	89.80	89.58	76.39	88.03	88.10	87.96	81.69	–	–	–	–

Table 3.2: SemEval-2015 results from the gold track (marked #), open track (marked *) and closed track (unmarked) of the in-domain (top) and out-of-domain (bottom). LF.av indicates the average LF score across all representations, and is used to rank the systems in their overall performance.

pose a sum over the substructures, and the highest scoring semantic graph is then chosen for a given sentence using dynamic programming.

A so-called **alternating directions dual decomposition** is used in order to approximate the highest scoring graph in order to reduce the search field. This reduces the original problem of finding the highest scoring dependency graph into sub-problems. These sub-problems are *predicate and arc-factored parts*, *unique roles*, *grandparents, arbitrary siblings and co-parents*, *predicate automata* and *argument automata*. For the technical details on training and parsing see: Martins and Almeida (2014).

3.4 Conclusions

In this chapter we have reviewed state-of-the-art semantic dependency parsers by examining a chosen set of parsing systems from SemEval-2015. Overall, we can state that our review demonstrates that the highest scoring systems are based on graph-based models rather than transition-based models among our chosen set. Tree-approximation models that use already well studied and highly developed syntactic dependency parsing systems can create highly accurate semantic depen-

dency parsers with an intermediary step where graphs are transformed to trees.

We will now perform an in-depth study of the results of three parsing systems among the ones presented in this chapter, namely the Peking, Turku and Lisbon parsing systems. This choice is grounded in the fact that these parsing systems have the highest accuracy among the SemEval-2015 submissions. The choice of disregarding the Riga system also stems from the fact that it is based on the Peking system, and the assumption that the type of errors that this system makes would resemble the Peking system seems legitimate. We now turn to our in-depth contrastive error analysis.

Chapter 4

In-depth Contrastive Error Analysis

In this chapter we present an in-depth contrastive error analysis of the results of SemEval-2015 for the Peking, Turku and Lisbon submissions. After examining several aspects of the results of the submissions, we present our most interesting findings, and use this as a basis for our experiments in the next two chapters.

The analysis of this chapter showed that for the purpose of our thesis an interesting task would be to examine semantic frame classification. Based on our analysis we will explore a set of features for training a semantic frame classifier, and setup an experiment which shows results of a hypothetical classifier that could be an extension of two of the parsing systems examined in this chapter: Lisbon and Peking.

Another interesting observation in our analysis is that singletons, i.e. nodes that are not attached to any other node in the semantic dependency graph, could potentially be an interesting case to study further. A binary classifier for predicting singletons could help improve the accuracy of semantic dependency parsing by using its output as additional features to the parser, or as a post-processing step after the parsing.

A description of the parsing systems that are part of our analysis can be found in Chapter 3. There where 6 submissions to SemEval-2015, including an 'unofficial' submission by a sub-set of the task organizers. We have made the choice of focusing on three of these parsing systems in our analysis. This is based on two criteria:

1. The chosen systems should be among those that produce the highest accuracy scores in SemEval-2015.

2. The technical approach of the three parsing systems should differ from one another: both the local transition-based and global graph-based models that we introduced in Chapter 2 should be represented. We can thus explore whether different technical approaches produce different types of errors.

The aim of our in-depth contrastive error analysis is to gain insights to the performance and errors that current state-of-the-art semantic dependency parsing systems make. The study is performed in order to:

1. Find similarities and differences in the results among a chosen set of parsing systems.
2. Compare and contrast their strengths and weaknesses.
3. Empirically identify and verify which types of errors that can be the focus of future research on improving the accuracy of semantic dependency parsing.
4. Examine the possibility of using the results of our study to modify and improve upon an existing system, or create a new system, in order to improve existing parsing systems.

In our analysis we draw inspiration from three similar studies made by McDonald and Nivre (2007), McDonald and Nivre (2011), and Choi, Tetreault, and Stent (2015). In these studies a comparative analysis of a set of syntactic parsers are presented, and various types of errors that these parsers produce are highlighted. The first and second study focus on three types of errors: (1) length factors, (2) graph factors, and (3) linguistic factors. The third study, in addition to these three factors, also examine the time complexity of parsing systems: both training and parsing time is taken into consideration. We will structure our analysis in a similar fashion, but exclude time complexity, and include the multi-classification task of semantic frame classification introduced in SemEval-2015. Our focus will thus be on four factors: (1) length factors, (2) graph factors, (3) linguistic factors, and (4) semantic frames.

In addition to narrowing down the scope in terms of choosing three parsing systems, we also exclude results for the PAS target representation. The reasoning behind this is that the DM and PAS target representations are relatively similar. Examining Table 3.1 in Chapter 3, we observe that DM and PAS are close to identical in the number of labels, percentage of graphs being trees, and percentage of dependencies being projective. The major difference between the two target representations is the percentage of so-called singletons, i.e. nodes not connected to any other node in the dependency graph. The DM target representation has approximately 5 times as many singletons as PAS.

With the exception of singletons, we assume that our analysis of the results for the DM target representation will yield similar results as for PAS. This hypothesis was confirmed by running the error analysis on the PAS target representation, and observing that for most types of errors there is a strong correlation in the performance of the parsing systems we have examined for DM and PAS.

		DM			PSD		
	LF.av	LF	LP	LR	LF	LP	LR
Peking	**85.33**	**89.09**	90.93	87.32	75.66	78.60	72.93
Lisbon	85.15	88.21	89.84	86.64	**76.36**	78.62	74.23
Lisbon*	**86.23**	**89.44**	90.52	88.39	**77.58**	79.88	75.41
Turku*	83.47	86.17	87.80	84.60	73.63	76.10	71.32

		DM			PSD		
	LF.av	LF	LP	LR	LF	LP	LR
Lisbon	**81.15**	81.75	84.81	78.90	**74.82**	78.68	71.31
Peking	80.78	**81.84**	84.29	79.53	73.28	77.36	69.61
Lisbon*	**82.53**	**83.77**	85.79	81.84	**76.18**	80.12	72.61
Turku*	78.85	79.01	81.54	76.63	71.59	74.92	68.55

Table 4.1: SemEval-2015 results from the closed track (unmarked) and open track (marked *) of the in-domain (top) and out-of-domain (bottom) data for the three parsers included our the analysis.

Before embarking on our in-depth contrastive error analysis, we will first recap and further examine some overall statistics on the three parsing systems that we will use examine in our study.

4.1 Overall Accuracy

In Chapter 3, we presented an overview of the technical aspects of the three parsing systems used for our analysis in this chapter. The Peking system: an ensemble of transition-based and graph-based models, the Turku system: a combination of several classifiers for classifying specific aspects of the semantic dependency graphs, and the Lisbon system: a graph-based feature-rich linear model that parametrize globally over first and second order dependencies.

In Table 4.1 we see data on the accuracy of the three parsing systems. The table has been reproduced from Oepen et al. (2015). The LF.av score is the averaged score across the three target representations: DM, PSD and PAS. We are interested in the LF scores as they relate to DM and PSD.

Examining Table 4.1, we observe that the Peking parser performs slightly better than Lisbon on average in the closed track. The Lisbon parser has a higher overall accuracy in the open track, which attests to the fact that using a syntactic parser as additional features for the parsing can increase the overall accuracy of semantic dependency parsing. However, The Peking parser has a higher accuracy in the closed track for PSD in comparison to Peking. We can conclude that the Lisbon and Peking systems show comparable results. An ensemble method where both these parsing systems are used may be an interesting case study for future work.

Even though the Turku parser participated in the open track, its overall accuracy is lower than that of Peking and Lisbon in the closed track. Another observation is that the Peking parsing system has a higher accuracy on the DM target representation, but lower than Lisbon on the PSD target representation. On the out-of-domain data we see that the Lisbon parsing system has the overall highest score.

It is worth noting that all three parsing systems have a substantially lower accuracy on the PSD target representation. One of the reasons for this lower overall accuracy on the PSD target representation can be attributed to the higher number of dependency types relative to DM. This is also true for semantic frames, where the PSD target representation has a higher number of frames.

4.1.1 Type of Errors

A somewhat obvious aspect of these results is that the parsing systems perform better on the in-domain versus the out-of-domain data sets. This is to be expected, as data-driven parsing will yield better results on data that is within the domain of the data used for training.

In terms of the specific types of errors we examine: length, graph and linguistic factors, there is an overall drop from in-domain to out-of-domain data, but we do not observe significant changes in the specific type of errors we have studied. So we expect the results of our analysis to yield similar results had we chosen to examine the out-of-domain data sets. The significant difference would have been an overall lower score for the factors examined. We do not prove this assumption, and more in-depth examination could yield results that contradict our observations.

We have, for the reasons mentioned above, chosen to solely focus on the results for the in-domain data. Our research has set out to explore the specific errors that different parsing systems make. We are therefore less interested in the errors that are related to factors that might be attributed to corpora, such as type of vocabulary, size of vocabulary, differences in sentence structure, and so forth. Such analysis would demand a different type of study where the corpora would be

subject to more in-depth analysis.

In SemEval-2015, the parsing could be run in an open and closed track. We refer the reader to Chapter 3 for details on the different tracks, and the approaches used by the parsing systems participating in the open tracks. Lisbon is the only team that participated in both tracks, Peking participated only in the closed track, and Turku only in the open track.[1]

For our analysis we have chosen to use data from the open track for Turku, since there are no data for Turku in the closed track, and the closed track for Lisbon and Peking. It is therefore important to note that the comparisons in our analysis must bear in mind that the Turku parsing system has the added benefit of using additional resources such as a syntactic parser: see Kanerva, Luotolahti, and Ginter (2015) for specific details on the additional resources used by Turku.

4.1.2 Measuring parsing accuracy

The measures used for determining the scores of the submission are *precision*, *recall*, and *F-score*. When calculating these, we use the measures *true positives*: instances that have been correctly predicted, *false positives*: instances that have been falsely identified, and *false negatives*: instances that should have been predicted, but have not been predicted.

Precision, also known as positive predictive value, is a measure for the *reliability* of a system's predictions. These are the dependencies that have been correctly assigned during parsing. We calculate this as follows:

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives + false positives}}$$

Recall, also known as sensitivity, is the measure for how *robust* a system is. These are the fraction of relevant dependencies that have been assigned during parsing. This measure is calculated as follows:

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives + false negatives}}$$

F-score is the so-called *harmonic mean* of the precision and recall. The harmonic mean is used as a way to weight the precision and recall of a system towards the lower end of both scores. This is done in order to create a balance between precision and recall so that we do not rely too heavily on either when determining the performance of a system. The F-score is used as the primary evaluation metric in our analysis of the results of SemEval-2015. It is calculated as follows:

[1] Turku also participated in the gold track, as the only team. We exclude the gold track from our analysis as it does not provide any significant insights to the comparative nature of our analysis.

$$\text{F-score} = 2 * \frac{\text{precision * recall}}{\text{precision + recall}}$$

We will now start our error analysis by first examining errors related to length factors, which for our purposes include sentence and dependency langths.

4.2 Length factors

As McDonald and Nivre (2007) point out, it is well known that syntactic parsing systems produce results with lower accuracy for longer sentences. We observe the same phenomena in the results of our three parsing systems: parsing accuracy has an inverse correlation with sentence length. As McDonald and Nivre (2011) observe, this is primarily due to more complex constructions in longer sentences, such as prepositions, conjunctions, and multi-class sentences.

Another type of length factor is the length of the dependencies themselves. We observe a decrease in parsing accuracy as the length of the dependencies increase. We define the length of a dependency from word w_i to w_j as $|j-i|$. In our analysis a length of 0 is used to denote a top node dependency. For the English language, and from examining the data sets used in SemEval-2015, we can generally state that short dependencies are modifiers of nouns, such as determiners, adjectives or pronouns. Longer dependencies are in most cases words that modify the main verb or root of the sentence.

4.2.1 Sentence length

In this section we will examine sentence length as a factor of parsing. Firstly, we point out the distribution of sentence lengths in the training and test data in Figure 4.1, and 4.2 respectively. The distribution approximates the *Bell curve*, and the average sentence length is 22.51 lexical units for the training, and 22.66 for the test data. However, the test data has a distribution where the approximation towards the Bell curve is more crude due to its relatively smaller size.

In Figures 4.3 and 4.4 we see the precision and recall for the three parsing systems for sentence length. The graphs in these Figures show precision and recall for sentences in bins of 10. If we look at the DM target representation, we see that the results of Lisbon and Peking are quite similar, where there is a correlation of precision and recall in relation to sentence length. The overall trend is that for longer sentences, the precision and recall of the parsing decreases. This overall trend was also observed by McDonald and Nivre (2007), McDonald and Nivre (2011), and Choi et al. (2015) when examining syntactic parsers.

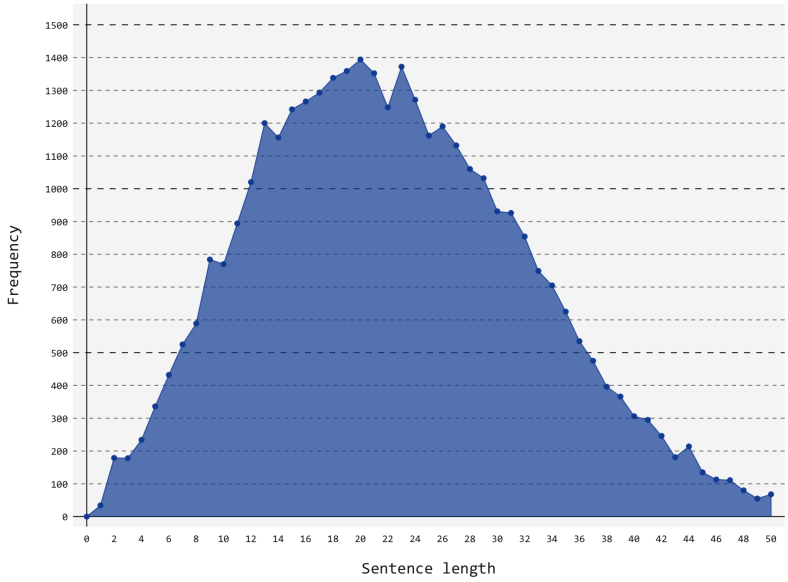

Figure 4.1: Distribution of sentence lengths and their frequency in the training data.

In comparison to the results from the syntactic dependency parsers examined in the studies of McDonald and Nivre (2007), McDonald and Nivre (2011), and Choi et al. (2015), in our analysis we see a slight upwards bump in precision and recall for the last bin that includes sentences that are longer than 50 lexical units. This can be explained by the fact that there are very few sentences with more than 50 lexical units in the testing data set, and that a slight bump might be attributed to the relatively smaller size of that bin, and thus an artifact. It is therefore worth noting that changing bin sizes would give us slightly different graphs, but that the overall trend would nonetheless be a slight decrease in accuracy as sentence lengths grow.

Another factor that can impact the bump in the last bin is that for semantic dependency graphs we might actually be dealing with two or more disjoint graphs. For sentences that have more than 50 lexical units, a sentence might produce a semantic dependency graph that would be similar to that of two sentences, and we thus would expect a higher accuracy on the combined accuracy of these two graphs for one long sentence. This would not be possible in syntactic dependency parsers, where we don't have the possibility of disjoint trees.

Studying the graphs in Figures 4.3 and 4.4, we observe that the Lisbon and Peking systems share quite a similar trajectory for both the DM and PSD target

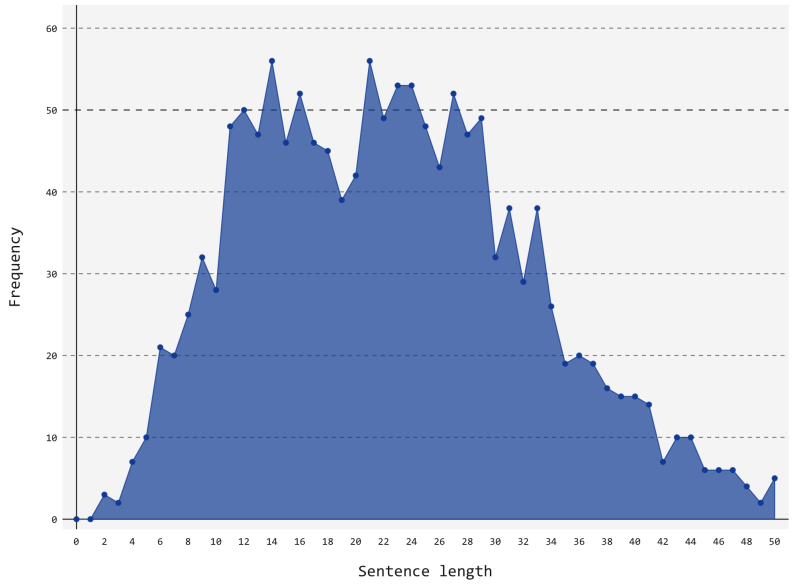

Figure 4.2: Distribution of sentence lengths and their frequency in the test data.

representations, with only subtle differences. The Peking parsing system performs better on the DM target representation, while the Lisbon parsing system performs better on the PSD target representation. However, for sentences smaller than 10 lexical units, the Peking parsing system has a higher precision and recall than Lisbon on the PSD target representation. On the DM target representation the trend is opposite; the Lisbon parsing system outperforms Peking on sentences that are longer than 50 lexical units. These difference might be attributed to the differences in the technical aspects of these parsing systems.

The Lisbon parsing system use a technique where several graphs are created, and then an approximation is used to select the highest scoring graph. For longer sentences this technique may result in higher accuracy as it may be more capable of producing the disjoint graphs that are possible for longer sentences. See Martins and Almeida (2014) for a detailed description of the Lisbon parser. The Peking system, on the other hand, use a transition-based model, and as such, longer sentences can be more challenging. This is due to the nature of transition-based models, where longer dependencies are more difficult to parse due to the linear nature of its transitions in the parsing.

Turku on the other hand has an overall different trajectory in comparison to the other two parsing systems on the DM target representation. The largest deviation from the overall trend is that the Turku system, when parsing on the DM target

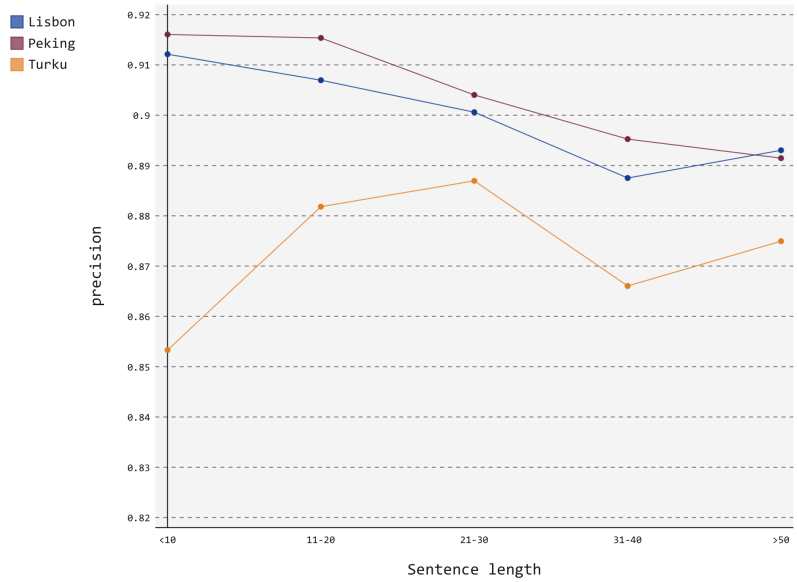

Figure 4.3: Precision relative to sentence length in bins of size 10. Precision for the DM target representation.

representation, show a particularly low precision and recall on sentences that have lower than 10 lexical units. This is not present when parsing on the PSD target representation, and should be considered an anomaly that might be attributed to some technical detail of the Turku parsing system. Since the Turku parsing system use a Support Vector Machine for its parsing, shorter sentences might pose an issue if the features used depend on more information than what is present for shorter sentences. More analysis is needed in order to clarify the reasoning behind this anomaly in the Turku parsing system.

4.2.2 Dependency length

Another interesting phenomena is the accuracy of a parsing system related to the length of dependencies. The parsing systems we examine have lower accuracy for longer dependencies. This is due to several reasons. One reason is sparsity, there are fewer longer dependencies than shorter. Another reason is that longer dependencies often represent more complex linguistic phenomena.

Observing the graphs in Figures 4.5, 4.6, we see two graphs representing the number of dependencies of each length for the Lisbon parsing system. The graphs have a distribution over the dependencies in the gold data, the predicted dependen-

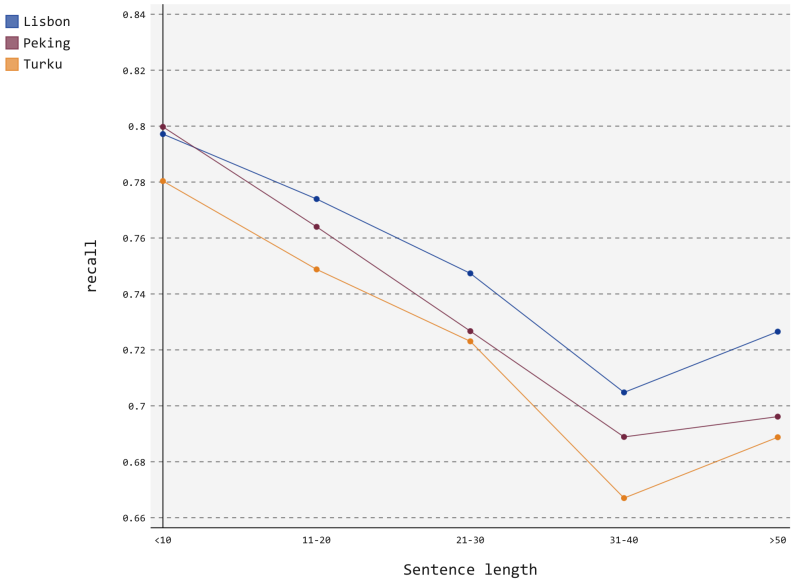

Figure 4.4: Recall relative to sentence length in bins of size 10. Recall for the PSD target representation.

cies, and the matches between the gold and predicted data (the correct predictions made by the parser).

The most obvious observation is that the Lisbon parsing system makes fewer predictions than what is found in the gold standard data set on both the DM and PSD target representations. The exception occurs when dealing with shorter dependencies, and examining dependencies of length 1 (a dependency between adjacent words), we observe that there are more dependencies in the predictions than gold standard data on both DM and PSD. As the length of dependencies increase, the number of predictions and matches decrease. These same observations hold for the Peking and Turku parsing systems.

To further elaborate on this point, we introduce the F-score of all parsing systems in relation to dependency length in Tables 4.7 and 4.8 for both target representations. Here we have divided dependency lengths in bins of 3 in order to make the graphs more readable. The F-score of the parsing systems for dependency length show a substantial decrease as dependency lengths increases.

The performance of Lisbon is noteworthy in this regard, and this may be part of the explanation as to why its overall performance is better than Peking and Turku. The Lisbon system has a substantially higher F-score for longer dependencies. Peking and Turku have somewhat similar results, with Peking having

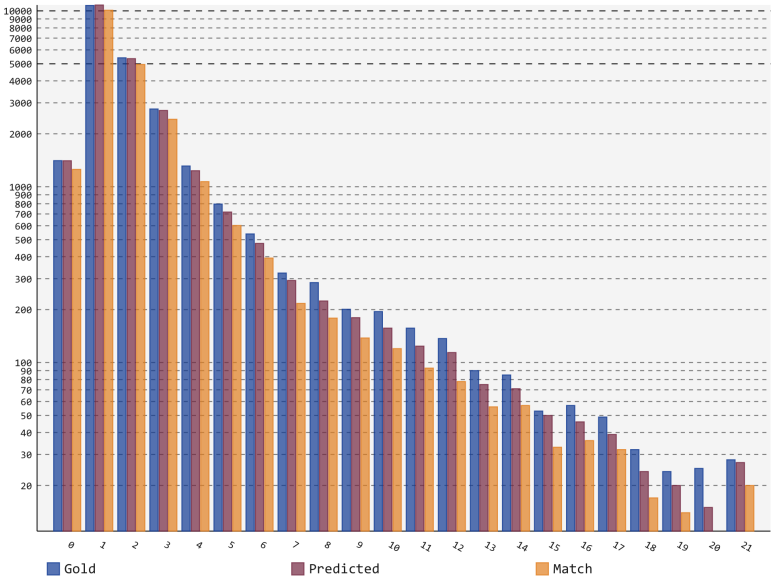

Figure 4.5: The number of dependencies for the Lisbon parsing system according to their length (where 0 denotes top nodes) for the DM target representation. The graph shows the number of dependencies in the gold data set, the predicted dependencies, and the matches between predicted and gold.

higher F-scores for dependency lengths that are between 1 and 12, but lower for dependencies above 12 (overall). This might be explained by the fact that Turku participated in the open track, and thus had access to other resources such as a syntactic parser. It might also be due to technical details of the parsing systems.

All three parsing systems demonstrate a decline in F-score as dependency lengths increase on both the DM and PSD target representations. However, there is a slight return to higher F-scores once the dependencies reach lengths ranging from 12 to 18. This is most prevalent for the Lisbon parser, but is also observed in the Turku and Peking parsing systems.

Both sentence and dependency length have a distribution in the data sets where the frequency of a length factor is correlated with the accuracy for parsing that specific length. Overall we can state that the higher the frequency of a given length factor, the more likely it is that the parsing system has a higher accuracy when parsing a sentence or dependency with a given length. It is therefore important not to exaggerate the importance of the length factor itself, but also take into account the distribution with which it occurs in the data used for training and testing. A different data set and distribution would result in, if our assumptions are correct,

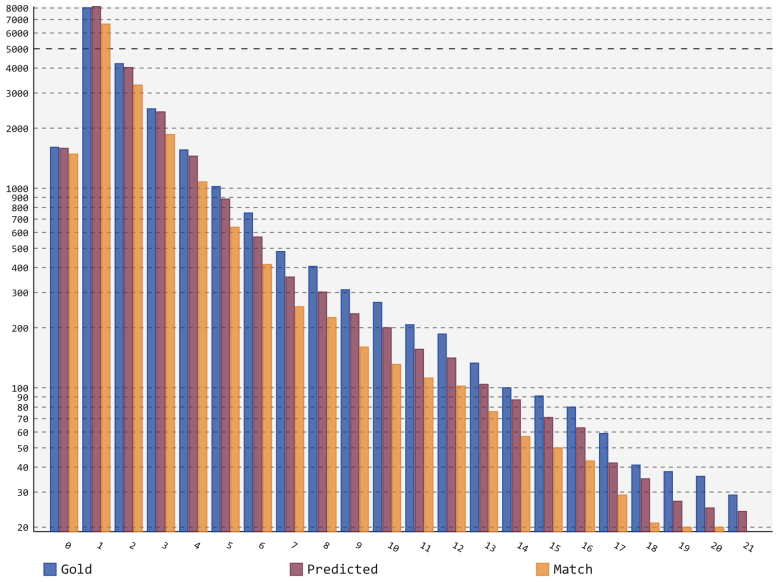

Figure 4.6: The number of dependencies for the Lisbon parsing system according to their length (where 0 denotes top nodes) for the PSD target representation. The graph shows the number of dependencies in the gold data set, the predicted dependencies, and the matches between predicted and gold.

parsing systems with a different correlation between length and accuracy.

However, since we are dealing with natural language, we can also assume that a relatively similar distribution of sentence and dependency lengths observed in the SemEval-2015 data sets will be prevalent in other corpora. It is therefore important for any research on dependency parsing to consider length factors. It is also important to further explore the linguistic factors that may impact the accuracy in relation to the length factors. Longer sentences include more complex sentence structure, and longer dependencies may include more complex grammatical structures.

We will now turn our attention to specific graph factors: singletons, and examine the accuracy of the three parsing systems in relation to these aspects of semantic dependency parsing.

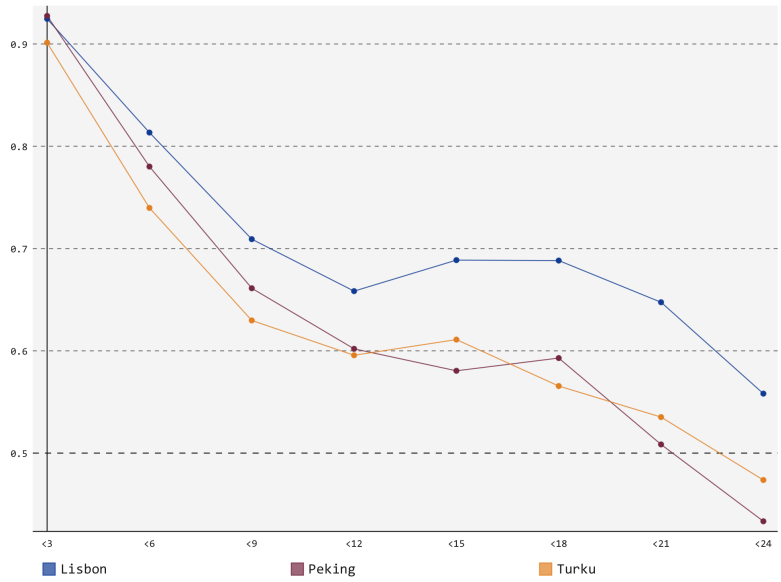

Figure 4.7: F-score for the three parsing systems on the DM target representations for dependency length in bins of 3.

4.3 Graph factors

As for graph factors we will examine the accuracy of the parsing systems of our analysis in relation to singletons, i.e. nodes that have been left outside of the dependency graph. This factor is specific to semantic dependency parsing, as with syntactic dependency parsing all lexical units are connected. In the semantic dependency graphs of the DM and PSD target representations, so-called semantically vacuous lexical units are left outside the dependency graph. We will examine how the three parsing systems perform in accurately predicting these.

4.3.1 Singletons

Examining Table 4.2, we can see the results of the three parsing systems in accurately predicting singletons. To be precise, the prediction of singletons in the three parsing systems are the result of nodes left out from the dependency structure rather. This is a step that could be considered as a binary classification task in its own right. However, the three parsing systems that we examine do not make any explicit classification of singletons, and the nodes left outside the graph are a side-effect of the parsing.

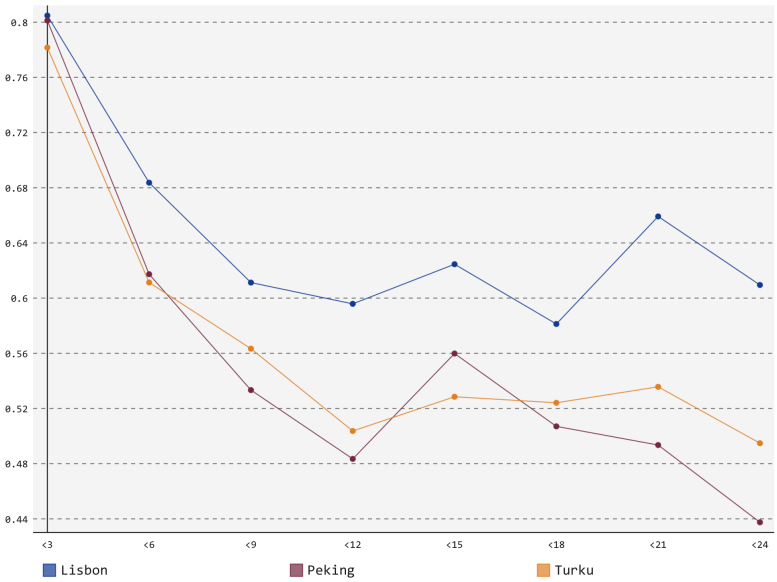

Figure 4.8: F-score for the three parsing systems on the PSD target representations for dependency length in bins of 3.

Looking closer at Figure 4.2, we see that for both target representation, Turku is the parser that has the highest number of singletons as the byproduct of its parsing, but a large portion of these should in fact be part of the dependency structure. The other two parsing systems also have a higher number of singletons than the gold standard. All parsing systems could therefore increase the accuracy of their parsing by attempting to parse more lexical units as part of the dependency structure.

A few cases do stand out regarding singletons. Examining Graphs 4.9 and 4.10, we observe that for certain part of speech tags, the accuracy is quite dramatically different than the others. The two graphs show the number of singletons, the predicted singleton by the parsing systems, and the match between these two, i.e. correctly established a lexical unit as a singleton.

For the DM target representation we observe substantial deviation in gold, predicted and match for 'VBZ' (verb, 3rd person singular present), 'VBD' (verb, past tense), 'POS' (possessive ending), 'RP' (particle) and 'RB' (adverb) tags. For the PSD target representation the 'RB' (adverb) tag stand out.

It is not clear as to why certain type of singletons are more difficult to parse than others, however, we suspect that it is connected to ambiguous words that have a wide range of different functions in a semantic dependency graph. Classifying

	Gold	Test	Match	Precision	Recall	F-score
Peking	7678	7681	7406	96.42	96.46	**96.44**
Lisbon	7678	7727	7425	96.09	96.70	96.40
Turku	7678	7850	7371	93.90	96.00	94.94
Peking	11600	11736	11508	98.06	99.21	**98.63**
Lisbon	11600	11912	11494	96.49	99.09	97.77
Turku	11600	12173	11474	94.26	98.91	96.53

Table 4.2: Results for singletons on the DM (top) and PSD (bottom) target representations.

singletons could be seen as a distinct binary classification task. Pursuing this task one could examine closely the specific type of singletons that are problematic for the parsing systems. A singleton classifier could then possibly be used as a pre-processing step before parsing, or as a post-processing step to clean up possible errors made by the parsing systems. However, since we have decided to pursue the task of semantic frame classification, we leave this task open for future work. The F-score of the systems for predicting singletons is also quite high, so it might be difficult to reach a more accurate system if we would pursue the task as a binary classification task.

4.4 Linguistic factors

The linguistic aspects of semantic dependency parsing that we will include in our analysis are dependency types. When dealing with dependency types, the analysis will differ to a higher degree between the target representations than other factors. This is due to the fact that the annotation schemes used in different target representations can be based on very different set of labels, and the number of unique labels can vary greatly. This is the case for the DM and PSD target representations. It is therefore important to note that the analysis in this section may not be as general as other findings such as the length factors.

4.4.1 Dependency types

The dependency types found in the DM and PSD target representations follow different annotations schemes. This is true for both the number of dependencies, their distribution and what they signify. In the test data set, the DM target rep-

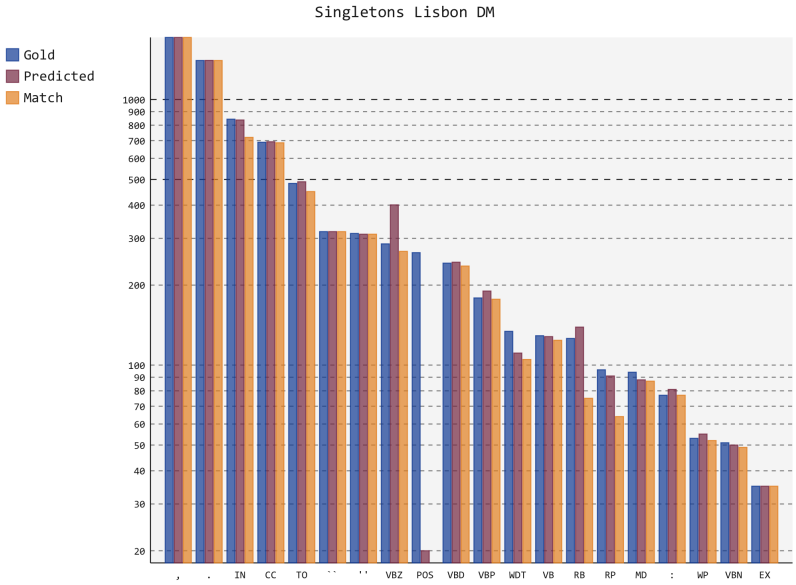

Figure 4.9: Singletons broken down by punctuation and part-of-speech tags for Lisbon on the DM target representation.

resentation has 43 dependency types and 24813 dependencies, whereas the PSD target representation has 77 types and 22258 dependencies.

If we examine Figures 4.11 and 4.12, we have two graphs with data for the 20 most frequent dependency types for both the DM and PSD target representation. The graphs present the number of dependencies in the gold data, the predictions made by the Lisbon parser, and the number of matches between the predicted and gold. We see that for the DM target representation the first two dependency types, 'ARG1' and 'ARG2', account for 14884 of the 24813 dependencies, while for the PSD target representation the distribution is more evenly spread across the most frequent types. In terms of the parsing accuracy, we see that the number of gold and predicted dependencies are more similar for DM than for PSD, where there are more fluctuations.

From the Figures 4.11 and 4.12, we observe that the parsing of the Lisbon parser is relatively conservative, i.e. there are less dependencies in the output than there are in the gold standard data. This is particularly evident for dependencies that are less frequent.

As we observed with singletons, there are a few dependency types that stand out with a larger margin of error. Examining Figure 4.11 we see that for Lisbon on the DM target representation, there is a great mismatch between the number

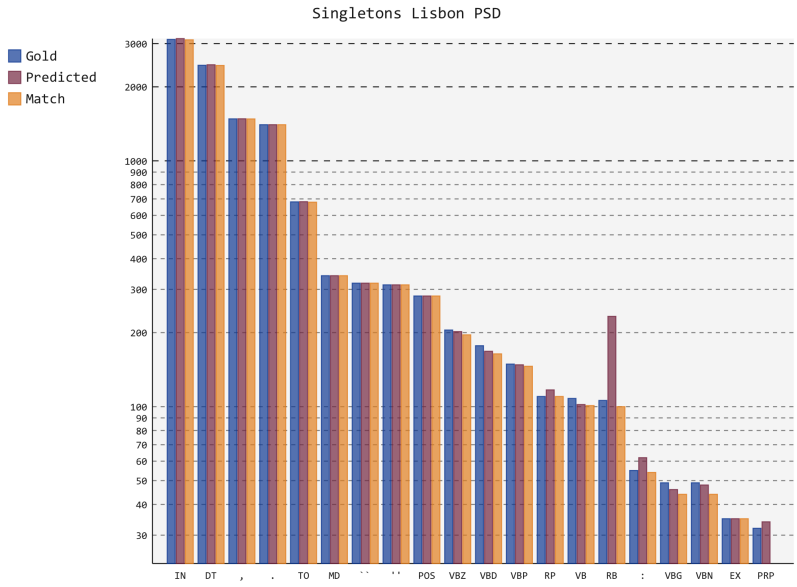

Figure 4.10: Singletons broken down by punctuation and part-of-speech tags for Lisbon on the PSD target representation.

of frames in the gold data, the predicted dependencies, and the match, for certain type of dependencies such as ‘conj’ (conjunction) and ‘subord’ (subordination). These are more complex grammatical structures than for instance ‘ARG1’ and ‘ARG2’.

In Figures 4.13 and 4.14 we see the F-scores of our parsers over the same distribution of the top 20 most frequent dependencies. There is a large discrepancy between the three parsers for each dependency type, indicating that the technical differences play a great role in this regard. The differences are less pronounced for the most frequent dependency types, but for ‘_and_c’, ‘ARG3’, ‘_or_c’, ‘part’, and ‘_but_c’, the variations are substantial. Although less extreme, we see the same tendencies for the PSD target representation.

An ensemble method of the semantic dependency parsing systems of our analysis might be an interesting approach if one is able to exploit the relative strengths of each system. An approach could be to take the output of all three parsers and by way of some heuristics choose the most probably dependencies based on the information of their respective error rates for each dependency type.

However, we would still face low scores for the type of dependencies where all three parsing systems have an equally low F-score. We also observe that for the most part, the Lisbon parser has a higher F-score on the individual dependency

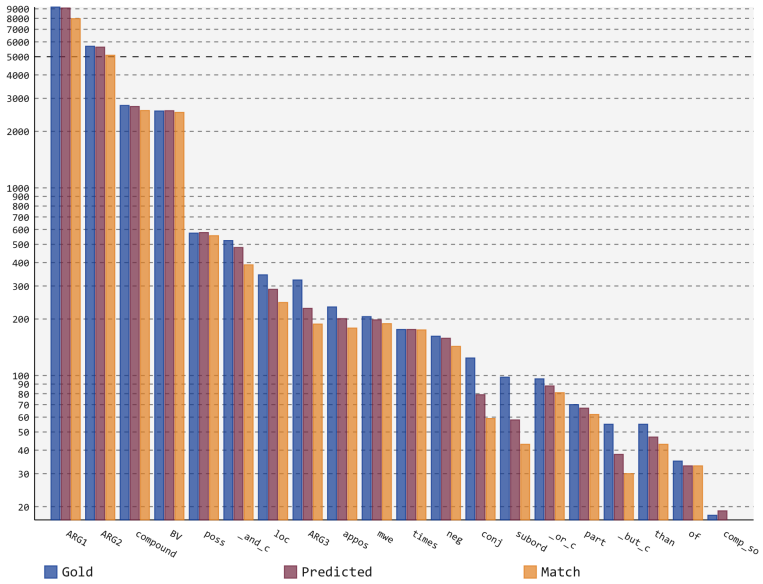

Figure 4.11: The 20 most frequent dependency types for Lisbon on the DM target representations. The graph shows the number of dependencies in the gold data set, the predicted dependencies, and the matches between predicted and gold.

types. An ensemble method would thus only boost the scores of those dependency types where the other parsers have a higher score. We will therefore not attempt such an approach as the results may not be any substantial increase compared to that of the Lisbon parsing system.

We now turn to the last aspect of our analysis, namely semantic frames. We will show that in relation to the other factors that we have examined in this chapter, semantic frames show the most potential for an increase in accuracy from the previous results.

4.5 Semantic Frames

Similar to dependency types, there is also a high degree of divergence between the two target representations that we are examining. In Table 4.3 and 4.4, we see some statistics on the frames in the data sets. It is important to note that we exclude a range of semantic frames in this study. This is done so that we may follow the scoring scheme set forth by the SemEval-2015 organizers, which the semantic frame scores for all the parsing systems are based upon.

Frame	Total	Percent	Frame*	Total*	Percent*
n:x	115049	18.64	n:x	4529	18.69
q:i-h-h	72278	11.71	q:i-h-h	2891	11.93
named:x-c	62208	10.08	named:x-c	2549	10.52
p:e-u-i	54513	8.83	p:e-u-i	2061	8.51
n_of:x-i	45542	7.38	n_of:x-i	1614	6.66
v:e-i-p	34475	5.59	v:e-i-p	1342	5.54
a:e-p	27802	4.5	a:e-p	988	4.08
card:i-i-c	27128	4.39	card:i-i-c	902	3.72
pron:x	14187	2.3	pron:x	647	2.67
n_of:x	12701	2.06	n_of:x	510	2.1
Total	465883	75.48	Total	18033	74.43

Table 4.3: The frequency distribution of frames in the training and test id (marked *) id for the DM target representation.

Frame	Total	Percent	Frame*	Total*	Percent*
ev-w218f2	8355	9.35	ev-w218f2	374	10.18
ev-w2833f1	7482	8.37	ev-w2833f1	350	9.53
ev-w1566f3	1731	1.94	ev-w1566f3	74	2.01
ev-w1239f1	845	0.95	ev-w2888f2	51	1.39
ev-w218f3	821	0.92	ev-w218f3	38	1.03
ev-w2888f2	810	0.91	ev-w410f1	31	0.84
ev-w2772f1	801	0.9	ev-w2875f4	29	0.79
ev-w218f7	657	0.73	ev-w1239f1	28	0.76
ev-w410f1	612	0.68	ev-w2671f1	24	0.65
ev-w3525f6	567	0.63	ev-w3586f1	22	0.6
Total	22681	25.37	Total	1021	27.8

Table 4.4: The frequency distribution of frames in the training and test id (marked *) id for the PSD target representation.

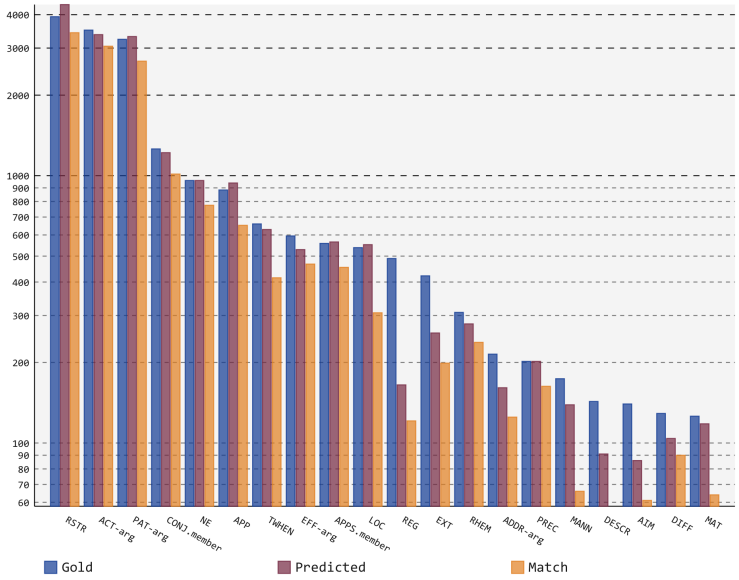

Figure 4.12: The 20 most frequent dependency types for Lisbon on the PSD target representations. The graph shows the number of dependencies in the gold data set, the predicted dependencies, and the matches between predicted and gold.

The SemEval-2015 scoring scheme reduces the number of frames to be part of evaluation by only including frames for tokens that have a *part of speech tag* that starts with the letter 'V'[2], and are classified as being predicates[3], i.e. a node with outgoing dependency edges in the semantic dependency graph. This was done in order to only score frames on potentially ambiguous verbs.

Both the number of unique frames and instances in the data sets have been reduced substantially by the scoring scheme, and the number of frames in DM and PSD are closer to each other. The DM and PSD in-domain data have 3459 and 3584 instances of frames to classify respectively, and 3750 and 3919 on the out-of-domain test data respectively. This makes it possible to have a more empirically similar base for comparing the results when classifying semantic frames on

[2]Part of speech tags that are equal to 'MD' should also have been included in this selection if the goal is to check for all possibly ambiguous verbs. Since we are interested in comparing our results to that of the SemEval-2015 submissions, we follow the scoring practices of the organizers.

[3]Some details on evaluation can be found here: `https://tinyurl.com/msh6jr8`. However, the specific details on the exclusion of frames not starting with 'V' and being a predicate are not present in any of the scientific papers that we have examined. The scoring scheme was revealed by examining the scoring files included in the package with the training and testing data

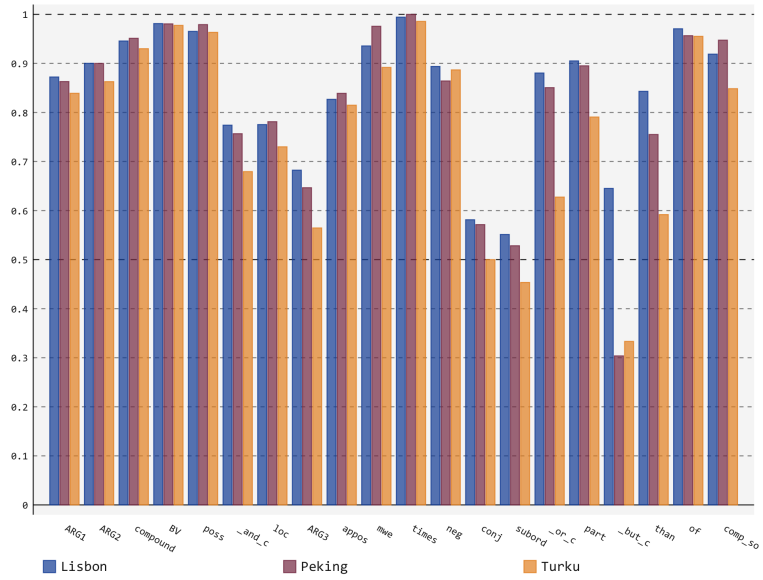

Figure 4.13: F-score for the three parsing systems for the 20 most frequent dependency types for the DM target representations.

both target representations. However, the number of unique frames are still somewhat skewed, where the unique number of frames in both training and testing are substantially higher for PSD in comparison to DM.

Examining Tables 4.3 and 4.4, we see the frequency and percentage of the top 10 most frequent frames in both target representations. It is important to note that the DM target representation has a much higher percentage of its total number of frames (75.48% in the training) in the top 10 in comparison with PSD (25.37% in the training). Another interesting phenomena to notice is that the distribution of frames in the training and test data for DM has a very similar distribution. The top 10 frames share approximately similar percentages, and have the exact same rank. For the PSD target representation we see that there are some variations on the top 10 frames. Some of the most frequent frames in the top ten for the training data does not appear in the testing data.

In Figure 4.15 we see a graph for the 30 most frequent frames in the DM target representation and the predictions made by the Peking parsing system. We observe that with the exception of a few frames, the Peking system is quite conservative, i.e. making fewer predictions than the gold data. As the frequency of the frames decrease, this gets more pronounced, and the number of correct matches also decrease substantially.

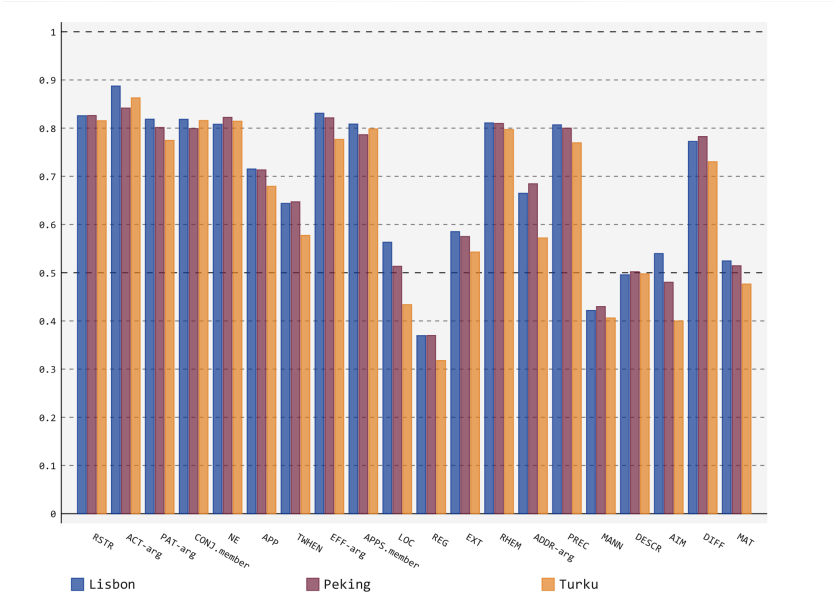

Figure 4.14: F-score for the three parsing systems for the 20 most frequent dependency types for the PSD target representations.

In Figure 4.16 we can observe the precision, recall, and F-score over the same distribution for the Peking system. The results are very interesting in the sense that there is quite a large fluctuation in scores for the 30 most frequent frames. There is a large discrepancy in precision and recall, at times very low scores, and as such we are left with low F-scores for a number of frames.

The fluctuations and relative distribution of precision, recall and F-score was also observed in Turku. The Lisbon team did not participate in the task of frame classification. The same type of numbers was also observed for the PSD target representation, i.e. quite similar graphs, and we therefore omit reporting the graphs visually.

From these observations we think that it would be interesting to examine frame classification as its own task. We will base the decision to do so on a solid empirical founding after having examined a wide variety of factors connected to the semantic dependency parsing systems of our study.

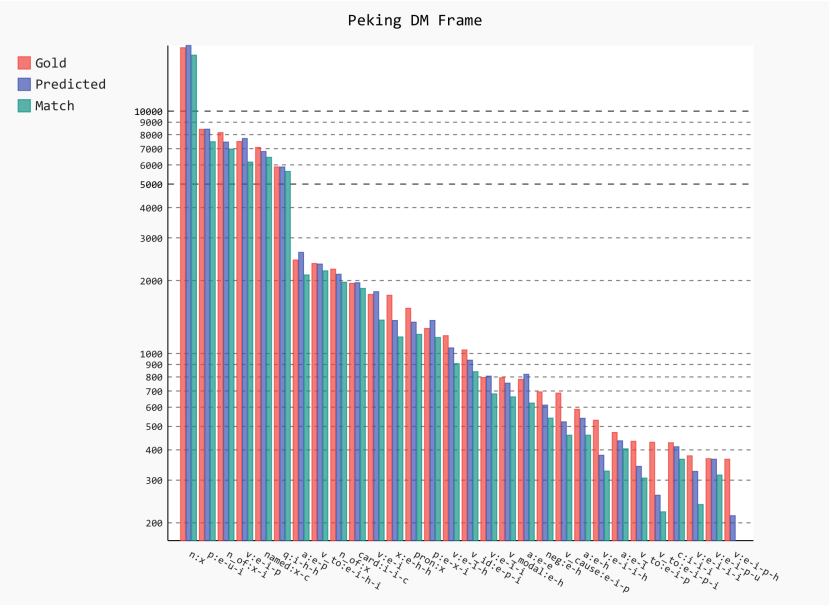

Figure 4.15: The distribution of the 30 most frequent semantic frames for Peking on the DM target representation. The graph shows the number of frames in the gold data set, the predicted frames, and the matches between predicted and gold.

4.6 Conclusions

In this chapter we have examined the results of Peking, Turku and Lisbon parsing systems as submitted to SemEval-2015. Our examinations have uncovered some interesting phenomena, such as the fact that the overall accuracy of all parsing systems drop in correlation to sentence and dependency length. That the parsing systems are more prone to errors in regards to certain graph and linguistic factors. We have shown that, although some parsing errors can be attributed to lower frequency in the training data, there are also other factors that impact parsing where an increase in training data would not necessarily translate to better results. These type of errors are more related to the technical aspects of the parsing systems themselves. We have seen that different parsing systems produce different types of errors. The results of such an overall analysis could lead the way for ensemble methods where an error analysis is used as basis for weighting.

However, the error analysis did not produce enough evidence to build an overall strategy for improving upon the results of the three semantic dependency parsers examined in this chapter. As mentioned previously, we have instead chosen to focus on improving the accuracy of the classification task of predicting

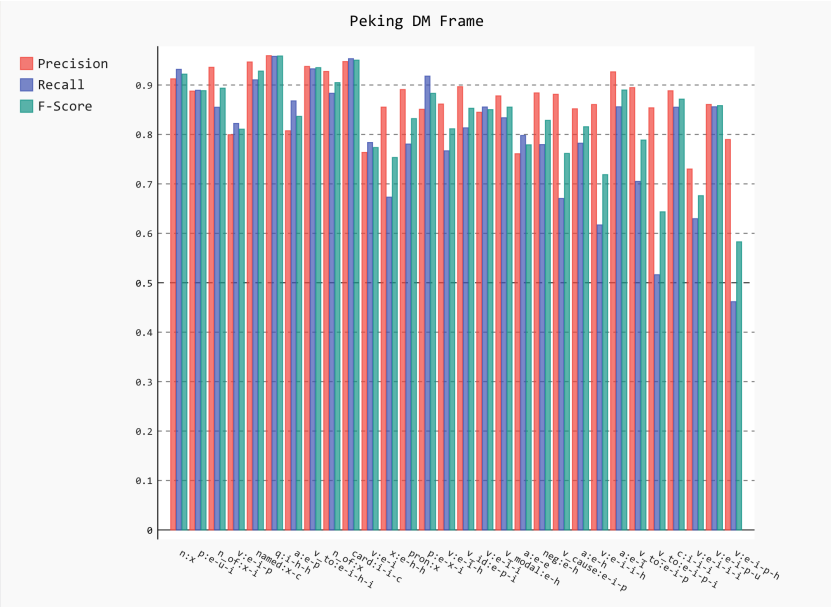

Figure 4.16: Precision, recall and F-score of the 30 most frequent semantic frames for Peking on the DM target representation.

semantic frames. This aspect of semantic dependency parsing is not well researched, and the results of our analysis seem to indicate that there can be room for improvement. In the next chapter we present a set of experiments where we attempt to improve upon the results of SemEval-2015.

Chapter 5

Semantic Frame Classification

The task that we embark upon is to build a *semantic frame classifier*. The steps involved in this process are threefold:

1. Find a machine learning algorithm for the classification task.
2. Investigate the effects of using different types of linguistically motivated features as input for training.
3. Present the results of our classifier and compare our results with the current state-of-the-art.

The motivating force behind our endeavours is the analysis presented in Chapter 4. Our explorations led us to the conclusion that an interesting aspect of semantic dependency parsing would be to elevate the accuracy of semantic frame classification. The potential that resides in a more accurate semantic frame classifier is the possibility of using its results to get better semantic dependency parsing results. The output of our semantic frame classifier could be used as additional features for a semantic dependency parser.

We start this chapter by taking a closer look at the data sets that we use for training, developing and testing, through a set of experiments, a classifier for predicting semantic frames. The experiments and results of the classification will be presented in Chapter 6. In this chapter we will focus on the different types of feature selection that we will examine, and the machine learning algorithms that we will use, as part of of our experiments in Chapter 6.

We use the DM and PSD target representations as basis for our experiments, and leave out PAS as it does not have semantic frames as part of its annotation scheme. This aligns with the analysis of the SemEval-2015 submissions from the previous chapter.

The main part of this chapter will be dedicated to presenting the *features* that we use as parameters for training our classifier. Features are individual properties that are either derived directly from the data sets, or by way of some transformation which can also include additional resources. We split our semantic frame classification into two different classifiers where the feature sets used for training differentiates their outcome and usage:

1. Classifying semantic frames based on *lexical*, *morphological* and *syntactic* features.

2. Classifying semantic frames based on *lexical*, *morphological* and *semantic* features.

The first set of experiments make use of features that do not rely on semantic dependency graphs. Our end result is a classifier where the predictions can be used as part of the input to a semantic dependency parser. This could possibly increase the accuracy of state-of-the-art semantic dependency parsers. The second set of experiments leads to a classifier that relies on the results of a semantic dependency parser for its classification. We leave out syntactic information in this set of experiments in order to compare our results to the SemEval-2015 submissions that participated in the closed track. The last set of experiments would be a semantic frame classifier that relies on both a syntactic and semantic parser, and would be comparable to the parsing systems that participated in the SemEval-2015 open track.

In this chapter we will also present our experimental setup, i.e. how we prepare the data for training, development and testing. We will also provide overviews of the machine learning algorithms used in our experiments. We have experimented with 4 machine learning algorithms: *Decision Trees*, *Support Vector Machines*, *Logistic Regression* and *K-nearest neighbors*. This ensures that we have a comparative basis for the choice of a machine learning algorithm for frame classification.

5.1 Experimental setup

We have already explored the details of our data sets in Chapter 3. In this section we focus on how we use the data to train, develop and evaluate our machine learning models. As part of SemEval-2015, an official test data set was made available, and as such we do not need to set aside a test set for our final evaluation. In order to have a data set for tuning while we are experimenting with our feature selection, we need to set aside part of the training data for development purposes.

Type	# Sentences	Frames	# Frames
DM training	28525	466	494122
DM development	7131	378	123128
DM test id	1410	290	24229
DM test ood	1849	299	23486
PSD training	28525	5074	72006
PSD development	7131	2705	17387
PSD test id	1410	1174	3673
PSD test ood	1849	1265	3882

Table 5.1: The data sets used for training, development and testing. The test set consists of in domain (id) and out of domain (ood) data. The columns signify number of sentences, number of unique frames, and number of occurences of frames.

Type	# Sentences	Frames	# Frames
DM training	28525	294	67493
DM development	7131	231	16295
DM test id	1410	162	3459
DM test ood	1849	173	3750
PSD training	28525	4951	69669
PSD development	7131	2634	16761
PSD test id	1410	1141	3584
PSD test ood	1849	1209	3919

Table 5.2: The data sets used for training, development and testing, but excluding frames on the basis of the rules of the SemEval-2015 evaluation criteria.

In Table 5.1 we can observe some higher order statistics on the training, development and test data sets. We have split the original training set using a commonly used 80-20 split, where we extract 20% of the training data for tuning and development purposes during our experimentation. This is to ensure that we do not run our experiments on the test data set, and thus avoid overfitting our machine learning models by selecting features that increase the accuracy of our models directly on the test data.

Once we are ready to run a final set of tests, we will train our classifiers on the whole training set, including the held out development set used for tuning purposes, and observe the accuracy of our models on the official held-out test set. This will then be presented as the final results of our classification task.

In Table 5.1 we observe that the test data consists of in-domain (id) and out-of-domain data (ood). We will test our machine learning models on both these data sets in the final rounds of testing. Our hypothesis, based on the results of the SemEval-2015 submissions, and general knowledge about machine learning algorithms, is that the accuracy of our classifier will be lower on the out-of-domain data sets.

Examining Table 5.1 further, we see that the training data consists of 28525 sentences, the development data set of 7131 sentences, the in-domain test set of 1410 sentences and out-of-domain test set of 1849 sentences. For the DM target representation we observe a total of 494122 occurrences of frames in the training set, consisting of 466 unique frames. For the PSD training set, we observe 72006 occurrences of frames and a total number of 5074 unique frames. It is therefore important to note that due to the higher number of unique frames in the PSD data set, we can hypothesize that we will observe a relative drop in the accuracy of our classifier in comparison to the DM target representation.

In evaluating the performance of our models, we follow the SemEval-2015 scoring scheme, which was presented in Chapter 4. The scoring scheme excludes all tokens that are singletons, and only evaluates those that have a part of speech tag that starts with the character 'V'. However, it is worth mentioning that that during training and prediction, we do not use this information. It is only at evaluation time that we exclude all the tokens that do not fit the scoring scheme.

Now that we have some further insight into the data, and the experimental setup for our classification task, we present the feature design that we will use in our experiments. It is worth noting that we are mainly interested in feature design in our experiments, and leave out parameter tuning of machine learning algorithms.

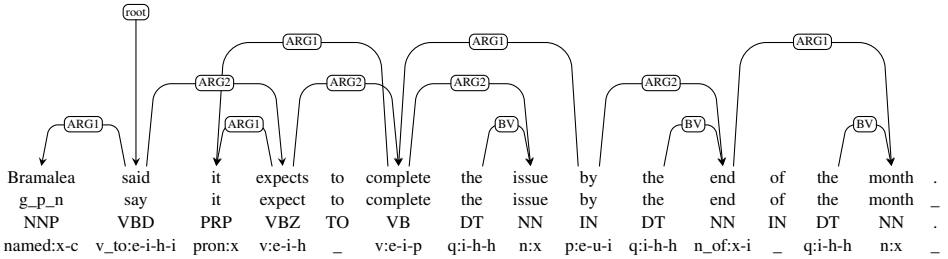

Figure 5.1: DM target representation with tokens, lemma, part of speech tags, semantic frames and labeled semantic dependencies.

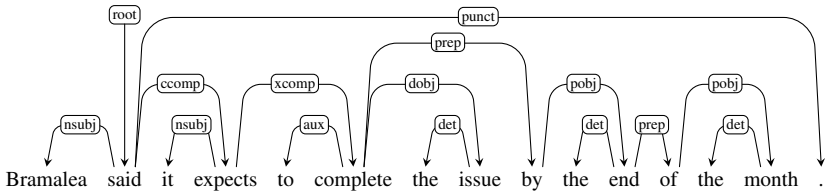

Figure 5.2: An example of syntactic dependencies, taken from the training data, and annotated with dependencies and labels with the so-called Stanford Basic scheme.

5.2 Feature Design

Feature design is the process of selecting the data that will be given to our machine learning algorithms as basis for its learning. For each token in the data, we extract a set of features that are used as input data for the machine learning algorithms as parameters for learning. During the training, each feature set has a corresponding semantic frame as its correct class. The machine learning will use this as basis for learning a mapping from a set of features to a correct semantic frame. Once a model has been trained, we can evaluate its performance by predicting classes for unseen tokens in the data used for development purposes. The score from this evaluation will be the basis for tuning, i.e. adjusting the set of features used for training.

There are no well defined methods by which we can empirically select the features that will result in the most accurate machine learning model. This is due to the fact that the feature space that is possible to construct is not finite, and we therefore have no way of testing all possible feature sets in order to find the set that will produce the most accurate classifier. We must therefore heuristically select our features by examining their impact on the accuracy of our classifiers.

When choosing features for our classifier we consider two factors:

1. Improving accuracy: We aim for features that increase overall accuracy of our model.

2. Reducing overfitting: We create a development set to ensure that we do not over-fit our features to the data.

We have chosen to base our feature extraction on 4 sets of *feature types*: lexical, morphological, syntactic and semantic.

Examining Figure 5.1, we have an example sentence taken from the training data which includes most of the information we will use as features. This example has been taken from the DM target representation. The first layer of information in Figure 5.1 is the labeled semantic dependencies. We will use these labeled dependencies as part of our feature space in the second classifier.

In the next layer of information we have the *token forms*, i.e. the words in their original form. The word forms are the units of information that are closest to the actual data (corpora), and the processing that has been performed in this layer is a tokenization step. For frame classification, we will use the token form as basis for our first set of experiments. However, as we will show in our experiments, using token form as our sole feature has its limitations, and with the size of our data and choice of machine learning algorithms, additional features are necessary in order to achieve state-of-the-art accuracy.

In the next layer of Figure 5.1, we have the *token lemma*. The *lemma* of a word is its base form representation. An example of this are the verbs 'run', 'runs', 'ran' and 'running', which are different word forms that share the same lemma 'run'. Lemmatisation is a *lossy* processing step whereby we lose some information by transforming words from one form to another. What we gain from processing tokens to find their base representation is a reduced vocabulary size, which in turn presents us with more coarse representations of a token and its context. The usage of lemma as features can thus increase classification accuracy for a set of tasks. However, some information is lost in this process, which for the case of semantic frame classification can lead to lower precision for tokens that are possibly mapped to a large set of semantic frames.

In the layer below the token lemma, we have part of speech tags. These are grammatical categories, such as nouns, verbs, and adverbs, that have been assigned to the tokens of a sentence. Part of speech tags can help in disambiguation tasks when there is a possible link to a semantic frame for a token that has a set of possible frames where token form, or token lemma, fall short.

In Figure 5.2 we have an example of syntactic dependencies. For the syntactic dependencies we use the companion files distributed as part of SemEval-2015,

where syntactic dependencies on the training and test data have been produced using the so-called Stanford Basic scheme derived from the Penn Treebank (Taylor, Marcus, & Santorini, 2003). The parser of Bohnet and Nivre (2012) have been used for the parsing. Syntactic features will be part of our first classifier.

We will now examine how we will use these features in our experiments, and give definitions for these feature sets.

5.2.1 Lexical and Morphological Features

The lexical features that we will examine in our experiments are the *token forms*, *token lemmas* themselves, and *prefixes* and *suffixes* that we can derive from these. The information contained in word forms are the most fine grained information that resides in a sentence, i.e. all other additional layers of information, such as lemmas, part of speech tags, syntactic and semantic dependencies, are in some ways an abstraction layer built upon the word forms themselves. However, for infrequent words, the word form is a sparse feature, and our machine learning models' predictive power will suffer as a result.

The lemma of a token will not fully resolve the sparsity connected to infrequent word forms. However, it will reduce the vocabulary of a corpus, and provide learning examples whereby a machine learning algorithm can connect an infrequent word form with its family of word forms, i.e. its lemma. However, for more frequent verbs, the disambiguation need to distinguish between many classes of semantic frames, and we thus have a risk of reducing the accuracy of our models. A combination of form and lemma, is more likely to produce greater accuracy, which will be confirmed through our experimentation.

Prefixes and suffixes of word forms may introduce patterns in the data that may be useful for semantic frame classification. Our hypothesis is that, particularly suffixes, with such patterns as 'ing', 'ed', and 'ang', may provide some additional information on the usage and context of a verb. However, these may also be too general too actually provide useful features for distinguishing between the correct semantic frame. We will examine the effects of adding prefixes and suffixes so that we can observe the potential effects such additional features may have. We compute prefixes and suffixes by extracting n characters from the end (for suffixes) and from the beginning (for prefixes) of a word form.

5.2.2 Syntactic Features

The syntactic features we will experiment with as part of our feature selection are the labeled dependencies in a syntactic dependency tree. Each sentence in the data set has a corresponding syntactic dependency tree. These trees are highly interesting in relation to our classification task. The semantic frame of a token

is in many cases connected to the number of arguments a verb can take and the type of dependency labels they have. Using the dependents of a token as a set of features may help distinguish between the various frames for ambiguous verbs where other features may be less relevant. Our experiments will include using the dependency labels in connection with the token form, token lemma, and the part of speech tag of the dependent.

We will also examine the impact that the head will have on detecting semantic frames. The head in a construct, as we described in Chapter 2, determines the semantic category of a construct, whereas the dependents gives semantic specification. Our hypothesis is therefore that heads are less informative features in distinguishing semantic frames. However, it is worthwhile examining this and empirically verify our hypothesis.

5.2.3 Semantic

For semantic features we are using the gold standard semantic graphs provided by the SemEval-2015 organizers in the DM and PSD target representations. In the same way as syntactic features, the semantic graphs can be useful in distinguishing frames by using the dependents in a semantic dependency graph. Examining Figure 5.1, we can illustrate this by examining the verb 'complete', which has two dependents: 'it' and 'issue', with the labels 'ARG1', and 'ARG2' respectively. This indicates that this particular verb can have two dependents that are of *type* 'ARG1' and 'ARG2', which might distinguish it from other verbs used in the same context.

When using dependents as features, we are faced with the possibility of having too sparse feature representations. Using the part of speech tags of dependents, instead of their form or lemma, can possibly help mitigate this, and lead to representations that are more informative for our classification.

Now that we have presented the possible feature space that we will use for our classification task, we will now examine the machine learning algorithms used as basis for our classification.

5.3 Classification Algorithms

We have chosen a set of four classifiers for predicting semantic frames in order to have a variety of approaches where we can empirically find a machine learning approach that yields satisfactory results. We start this chapter by providing a short description of the four algorithms that we have chosen for our classification task. These four machine learning algorithms are all implemented and openly available as part of the *scikit-learn toolkit* (Pedregosa et al., 2011).

Scikit-learn is a high-level machine learning toolkit written in the Python programming language. The authors of scikit-learn describe the toolkit as a set of state-of-the art machine learning algorithms that have been designed for usage on medium-scale supervised and unsupervised problems (Pedregosa et al., 2011). It is a well established machine learning toolkit that is both used in research and the industry.

It is worth noting that we ran the experiments on all four algorithms to a certain limit. Once we established the classifier that consistently performed with the highest accuracy on our chosen set of initial features, we limited further explorations to the most accurate algorithm. Since running the experiments on each classifier is time consuming, this decision was necessary once we started performing more in-depth feature selections so that we could proceed in a timely fashion. Our aim is to examine the possibilities of increasing the accuracy of frame detection to rival that of the current state-of-the-art. See Chapter 3 for an overview of the most accuracy semantic frame classifiers.

5.3.1 Decision Trees

Decision Tree Classifiers learn rules by creating a tree structure with simple rules that are easy to analyze and understand. This can be contrasted to more complex machine learning algorithms such as *neural networks*. In the latter cases the learning and predictions made by the algorithm acts like a 'black-box' in terms of comprehensibility.

The decision tree algorithm learns based on a set of labeled instances by inductively setting up a set of rules which form a tree structure. The tree structure can be visualized by thinking of each node in the tree as a question. Classification is done by starting at the root node, and based on the rules that have been learned, classify new data by traversing the tree and reaching a node that is designated by a specific class. For each node, a check is made on the unseen data that is to be labeled as the traversing step. As Kotsiantis (2013) notes, the process by which a decision tree classifier makes predictions is similar to a greedy search.

The decision tree classifier that is part of the scikit-learn toolkit has a small set of options for creating different types of decision trees. The options include setting the split criteria, where the options are *gini* and *entropy*. As Kotsiantis note, gini and entropy are two different measures used as the splitting measure that the learning algorithm uses in order to create the decision trees (Kotsiantis, 2013).

Our experiments showed that decision trees can be a fast and accurate machine learning algorithm for frame classification. However, it did not score high enough to be considered for further exploration beyond the morphological features described in Section 5.2. We also observed that the high dimensionality of features

used in our classifier was not suited for a decision tree classifier.

5.3.2 Support Vector Machines

Support vector machines are well suited for multi-class classification tasks, particularly when dealing with high dimensional feature sets as in our case. As Hsu and Lin (2002) note, at the time when support vector machines where created it was initially made for binary classification. However, a number of methods have been constructed whereby a support vector machine can be extended to handle multiple classes. Support vector machines take a set of instance-label pairs, and solves an unconstrained optimization problem by way of different loss functions. We will not go into the mathematical details of support vector machines, and the reader is referred to (Fan, Chang, Hsieh, Wang, & Lin, 2008) for details on the theoretical foundations, and the actual implementation of this algorithm that we will use.

The scikit-learn toolkit has 3 implementations of support vector machines. After a few experiments, we ended up using the 'LinearSVC' class, which is a wrapper around the 'LIBLINEAR' library developed by Fan et al. (2008). Without parameter tuning, this implementation proved the most accurate on our initial set of experiments. We do not include the experiments run on the other implementations that are part of the scikit-learn tookit.

The 'LIBLINEAR' library is an open source library for large-scale linear classification. It is well suited for large data and feature sets, and it is particularly recommended for text classification by its developers. In certain cases it is also known to be faster than many support vector implementations, including the often used 'LIBSVM' library (Fan et al., 2008). The other scikit-learn support vector machine algorithms are based on 'LIBSVM', which proved to be relatively slower once our feature sets grew to include syntactic and semantic features. We found that by using the 'LinearSVC' class in the scikit-learn toolkit we reduce training time, which made it possible to run more experiments in comparison with the other implementations.

Our experiments showed that support vector machines had the highest overall scores among the four machine learning algorithms on the first set of morphological features. It is a versatile algorithm that performed well without any parameter tuning. We therefore opted for this algorithm once we reached a threshold of experimentation on the basis of its comparatively higher accuracy. It is not definite that the other machine learning algorithms could not produce comparable, or even higher accuracy. However, this would have demanded a greater effort in examining the possible range of parameter tuning for all four algorithms, which unfortunately was deemed outside the scope of our thesis.

5.3.3 Logistic Regression

Logistic regression is a machine learning model based on regression analysis. The implementation used in scikit-learn, like the support vector machine implementation described above, is based on the 'LIBLINEAR' library. The implementation is of a multivariate logistic regression model with the loss function being $log(1 + e^{y_i w^T xi})$, which has been derived from a probabilistic model (Fan et al., 2008).

We achieved relatively high accuracy using the logistic regression model at a training time that allowed for experimentation with different feature sets. However, as we saw that the support vector implementation scored relatively higher for all lexical and morphological features, we did not further test this algorithm once we started experimenting with syntactic and semantic features. It is our hypothesis that with devoted analysis to the effects of parameter tuning, logistic regression could potentially be an interesting rival to support vector machines for our case. This is based on the fact that logistic regression and support vector machine implementations showed relatively similar accuracy levels on most of the experiments we ran using both algorithms.

5.3.4 K-nearest neighbors

Nearest neighbors classification is a type of machine learning method where training data is stored directly in a model, and a computation is performed by way of a simple majority vote of the nearest neighbors of each point in the model. It is a non-parametric method used for both classification and regression, wherein the classification the output of our model is a class, and in regression the output is the average of the values of its *k* neighbors (Altman, 1992). The k-nearest neighbors algorithm can also be used as an unsupervised learning algorithm, and is often used as basis for tasks that involve some form of clustering.

We did not achieve comparably high accuracy using the k-nearest neighbors classifier implemented in the scikit-learn toolkit. We therefore opted to leave out this algorithm as well once we started our experiments with syntactic and semantic features. In fact, for high feature dimensions, the accuracy of this algorithm performed well below the other three algorithms we examined.

5.4 Conclusions

In this chapter we have reviewed our experimental setup. We have presented the training, development, and test data, and given an overview of the distribution of semantic frames across these sets. We have also presented the feature types that

we will use as basis for our experiments, and the machine learning algorithms that we will examine as basis for creating our classifiers. In the next chapter we will present the results of our feature selection and experiments with different machine learning techniques. We show that with in-depth feature selection, it is possible to rival the accuracy of state-of-the-art semantic frame classification, and that our results can be used as basis for enhancing the best submissions in the closed track of SemEval-2015.

Chapter 6

Experiments

In this chapter we will examine and present experimental results for the task of frame classification. We will use the data sets presented in Chapter 4, and 5 for training, development and testing. We will examine how each set of features impacts the precision, recall and F-score of our models. The mathematical definitions of these measures can be found in Chapter 4. The end result of our experiments will be two different semantic frame classifiers based on the feature sets used for training. Both classifiers share the same lexical and morphological features, but diverge in using syntactic and semantic dependencies as additional features on top of these. Once we have finished our experiments on the development data set, we present our final results on the test data. Our results show that with rigorous experimentation with different combination of features, we were able to obtain state-of-the-art results for semantic frame classification. We compare our results with previous results in frame classification. We also show that by extending two of the participating systems of SemEval-2015, namely Lisbon and Peking, our results can be used to improve these system's semantic frame classification accuracy.

6.1 Baseline

Before we start our experiments, we need to define a baseline in order to have a common denominator that we can evaluate our results against. There are no agreed upon standards for defining a baseline for classification tasks such as semantic frame classification. Statistical counting measures have been the standard in other tasks, such as *part of speech tagging*. We will draw inspiration from methods used in part of speech tagging for setting up a baseline for our task.

An often used baseline in part of speech tagging is a *most frequent class* approach: given an ambiguous word, assign to it the most frequent part of speech

Representation	Precision	Recall	F-score
DM	72.09	73.70	71.76
PSD	58.63	66.55	61.47

Table 6.1: Baseline score with a most frequent frame per lemma approach.

tag (Jurafsky & Martin, 2009). In part of speech tagging most words are unambiguous (80-86%), and we are therefore left with approximately 14-15% of words where using the most frequent tag for for said word will results in possible errors. Running a most frequent class baseline classifier for part of speech tagging on the WSJ corpus results in an accuracy of 92.34% (Jurafsky & Martin, 2009). State-of-the art part of speech taggers have been able to achieve accuracy scores above 97%, which is a significant increase from the baseline.

Examining Tables 4.3 and 4.4 in Chapter 4, we observe that the ten most frequent frames in both data sets account for 18.64% and 9.35% of all frames respectively. Given the number of frames and their distribution, we will use the same approach as part of speech tagging and use a *most frequent class per lemma* approach for our baseline. For each lemma in the development set, we assign to it the most frequent frame encountered for that lemma in the training set. For lemmas that have not been encountered in the training, we assign the overall *most frequent frame* in the training data. This approach results in a *strong* baseline. A *weak* baseline would be a baseline where we assigned the most frequent frame to all tokens. Another option, which would also be a weak baseline, is a *random* baseline where we randomly assign frames to each token.

In Table 6.1, we have listed the baselines using the most frequent frame per lemma approach. It is worth reminding that we evaluate the baseline, as with all experiments in this chapter, by only including tokens that have a part of speech tag that starts with the character 'V' and is not a singleton. However, we can only access the part of speech tag information during training and prediction, and it is only after classification that we use information that a token is a singleton or not, based on the gold standard data, for scoring. A detailed description of the scoring scheme is described in Chapter 4.

We observe that the baseline for DM is significantly higher than PSD. We explain this discrepancy by noting that the PSD target representation consists of a much larger set of frames. If we examine frequent lemmas, such as 'be', 'make', 'have', 'take', 'say', we observe that for these verbs, the number of frames assigned in the PSD target representations is higher, and the distribution of the frames assigned are more uniform. A most frequent frame per lemma approach

	Classifier	Precision	Recall	F-score
DM	Baseline	72.09	73.70	**71.76**
	Support Vector Machine	70.85	70.98	69.58
	Decision Tree	71.05	70.89	69.60
	Logistic Regression	67.04	68.00	65.34
	K-Nearest Neighbor	69.32	63.44	65.00
PSD	Baseline	58.63	66.55	61.47
	Support Vector Machine	66.91	66.88	**64.83**
	Decision Tree	66.79	66.76	64.72
	Logistic Regression	62.11	60.57	59.33
	K-Nearest Neighbor	64.34	64.10	62.74

Table 6.2: Results for *form* as the sole feature.

will therefore be less effective for the PSD target representation.

Now that we have established a baseline we start with our first set of experiments using *lexical* and *morphological* features for training. We then go on to examining *syntactic* dependencies as features, resulting in our first semantic frame classifier. After syntactic features we will introduce the results of using *semantic* dependencies as features, resulting in our second classifier.

6.2 Experiments

We start things off by examining a set of features in order to establish some basic information on the results we can obtain on the four machine learning algorithms we will examine. With this information at hand we choose one of these four algorithms to continue our experiments with. We start by examining lexical features and observe the scores we can obtain by solely using these so-called surface features.

6.2.1 Lexical Features

Token Form The first feature we will examine is the *form* of verb tokens. For each token that is a verb, we use its form as the sole feature for classifying the frame of that token. In Table 6.2 we have listed the results using this feature on the four machine learning algorithms we are experimenting with. It is interesting to note that for the PSD target representation we achieve higher F-score than the

	Classifier	Precision	Recall	F-score
DM	Baseline	72.09	73.70	71.76
	Support Vector Machine	71.79	74.20	**71.81**
	Decision Tree	71.67	74.12	71.74
	Logistic Regression	69.97	72.98	70.00
	K-Nearest Neighbor	69.61	68.18	67.60
PSD	Baseline	58.63	66.55	**61.47**
	Support Vector Machine	58.55	66.47	61.39
	Decision Tree	58.60	66.52	61.44
	Logistic Regression	55.43	63.22	58.23
	K-Nearest Neighbor	55.69	63.30	58.40

Table 6.3: Results for *lemma* as the sole feature

	Classifier	Precision	Recall	F-score
DM	Baseline	72.09	73.70	71.76
	Support Vector Machine	75.67	74.01	**72.68**
	Decision Tree	75.80	73.78	72.48
	Logistic Regression	71.35	72.93	71.00
	K-Nearest Neighbor	74.96	72.55	72.23
PSD	Baseline	58.63	66.55	61.47
	Support Vector Machine	70.34	70.86	**68.62**
	Decision Tree	69.86	70.10	68.02
	Logistic Regression	67.66	68.55	66.20
	K-Nearest Neighbor	66.97	68.70	66.36

Table 6.4: Results for *form* and *lemma* as features

	Classifier	Precision	Recall	F-score
DM	Baseline	72.09	73.70	71.76
	Support Vector Machine	85.20	84.44	**84.10**
	Decision Tree	78.49	76.90	76.73
	Logistic Regression	82.22	80.69	79.54
	K-Nearest Neighbor	64.33	55.83	57.45
PSD	Baseline	58.63	66.55	61.47
	Support Vector Machine	80.09	80.38	**79.09**
	Decision Tree	77.51	74.19	74.53
	Logistic Regression	71.31	71.90	69.85
	K-Nearest Neighbor	44.14	37.05	37.24

Table 6.5: Results for *form* and *lemma* on the main verb token, and a context window of $n = 3$ where we use *token form* as the window.

baseline on all machine learning algorithms with the exception of Logistic Regression. However, with the exception of Support Vector Machines, we also observe that we achieve lower recall than the baseline. This might be due to the relative uniform distribution of frames on ambiguous verbs in PSD, and as such a most frequent frame per lemma baseline will have a higher recall than precision. In comparison, we observe that the difference between precision and recall is much higher for PSD than DM.

The scores for the DM target representation are approximately 3 to 7 percentage points lower than the baseline. This is due to the fact that we are using the tokens form, as opposed to the lemma, which is what we have used in our baseline.

Token Lemma In our next run we will examine how using *lemma* as the sole feature, in the same way as in our baseline, impacts the learning. The results are presented in Table 6.3. We observe that the results are overall higher than using form as our sole feature for the DM target representation, but lower for PSD. Our hypothesis is that this is, as with our baseline score, an artifact of the number and distribution of frames across the target representations.

Combination of Form and Lemma Let us now turn to combining lexical features. We start by examining whether the combination of form and lemma results in a higher score in comparison to using form or lemma in isolation. In Table 6.4 we see that we achieve a slight increase for the DM target representation when

compared to the results using just lemma, but a significant increase in PSD. The combination of the form and lemma of a token will has the information needed to achieve higher scores than the baseline.

Context Window For our next experiment we examine how the context window of a token, referred to as *token n-grams*, might impact the accuracy of our machine learning models. We will present experiments using a context window of 3 tokens surrounding the main token: 'form-3', 'form-2', 'form-1', 'form', 'form+1', 'form+2', 'form+3'. The words are added as individual features using a bag-of-words approach, so order does not have an impact. This is done so that we avoid too sparse vectors that would have been the results of using concatenated tokens as features. Had we concatenated the words so that the word order was kept intact, we would end up with very infrequent strings, i.e. sparse features. In Table 6.5 we see the results of our run with a context window of $n = 3$, meaning that we have used 3 tokens to the left, and 3 tokens to the right, of our main verb token.

For the single features that we experimented with previously, the differences in results between the machine learning algorithms were less obvious. However, once we use a combination of features we observe that there is a higher degree of divergence. The Support Vector Machine algorithm achieves the highest score for both DM and PSD when using a context window by a significant margin. If we look at Table 6.5, we see that the F-score for DM for Support Vector Machine is 84.10%, an increase from an F-score of 72.68% in the previous experiments where we used form and lemma of the main token. For PSD we see an increase from an F-score of 68.62% to 79.09%. It is interesting to note that for the K-Nearest Neighbor Classifier we observed a decrease in score in comparison with previous results.

The lower scores for the other algorithms was observed on all the experiments that are presented in this chapter. We therefore leave out these results from our presentation in order to keep the tables and presentation more compact and readable. However, it is worth noting that parameter tuning has not been performed in this study. It is therefore not possible to empirically argue that our choice in using Support Vector Machines leads to the most accurate classifier. For that we would need to perform an in-depth comparative study where we performed a range of tuning exercises for each algorithm. However, we deemed this outside the scope of our thesis.

Increased Context Window Let us now increase the context window with a larger n, and use both *form* and *lemma* as basis for the surrounding n elements. We will examine their performance both individually and in conjunction. Our context window for our experiments will be $\{n|n > 2 \wedge n < 8\}$. Tables 6.6, 6.7, and 6.8,

Representation	N-gram	Precision	Recall	F-score
DM	$n = 3$	85.20	84.44	84.10
DM	$n = 4$	87.17	86.52	**86.32**
DM	$n = 5$	86.88	86.20	85.99
PSD	$n = 3$	80.09	80.38	79.09
PSD	$n = 4$	83.20	83.66	**82.41**
PSD	$n = 5$	82.54	83.09	81.73

Table 6.6: Results for experiments with context windows of $\{n|n > 2 \wedge n < 6\}$ using only *form* as the context window.

show the results for $\{n|n > 2 \wedge n < 6\}$, for form, lemma and both respectively.

Table 6.6 shows the results for a context window consisting of form. We see an increase in F-score from $n = 3$ to $n = 4$, for both the DM and PSD target representations. We then observe a decrease once we reach $n = 5$ for the context window. This decrease continued as n increase. We omit these scores so that our tables are more readable.

In Table 6.7 we observe a similar trend. The best results are for $n = 3$, and we observe a higher score in comparison to Table 6.6 where we used form in our context window. When we combine form and lemma, as seen in Table 6.8, we do not get higher scores than using just lemma.

For the context window, we observe that the best feature set for our task is using lemma with a context window of $n = 3$. For our experiments going forward this will be the base configuration. The results for this set of features are an F-score of 86.58% for DM and 83.20% for PSD. These are the highest scores achieved thus far. We now turn to syntactic dependencies and examine their impact on the accuracy of our system.

6.2.2 Morphological Features

The morphological features included in our experiments are part of speech tags, suffixes and prefixes. We will examine their impact on the accuracy of our classifier by examining each individually, and then adding the features that have the highest impact in a similar fashion as the lexical experiments.

Part of Speech Tags with Form and Lemma We will now examine the effects of adding part of speech tags to the form and lemma tokens. We will concatenate

Representation	N-gram	Precision	Recall	F-score
DM	$n = 3$	87.34	86.67	**86.58**
DM	$n = 4$	87.07	86.46	86.32
DM	$n = 5$	86.7	86.12	85.97
PSD	$n = 3$	83.88	84.39	**83.20**
PSD	$n = 4$	83.55	84.07	82.79
PSD	$n = 5$	83.01	83.61	82.25

Table 6.7: Results for experiments with context windows of $\{n|n > 2 \wedge n < 6\}$ using only *lemma* as the context window.

Representation	N-gram	Precision	Recall	F-score
DM	$n = 3$	87.08	86.43	**86.29**
DM	$n = 4$	86.51	86.05	85.79
DM	$n = 5$	86.34	85.86	85.59
PSD	$n = 3$	83.39	83.98	**82.71**
PSD	$n = 4$	82.80	83.48	82.10
PSD	$n = 5$	82.37	83.26	81.81

Table 6.8: Results for experiments with context windows of $\{n|n > 2 \wedge n < 6\}$ using *lemma* and *form* as the context window.

the tags at the end of each token: <form_pos> and <lemma_pos>. The results of our experiments are presented in Table 6.9. In the first experiment, we concatenate the part of speech tag to the form token, leaving lemma as a feature on its own. We then do the opposite, concatenating the part of speech tag to the lemma, but leaving form as its own feature. We then concatenate the part of speech tag to both form and lemma. As a final run we combine all of the above: form and lemma as their own features, form and lemma concatenated with part of speech tag, and part of speech tag as its own separate feature.

From Table 6.9, we observe that the various strategies of using part of speech tags yield similar results. We see that the two highest scores, for both DM and PSD, are when we either concatenate part of speech tags to form, and leave lemma as a separate feature, or when we leave both form and lemma as a separate features, and the part of speech tag as its own separate feature. However, overall, the increase in precision, recall and F-score are relatively low in both cases, where for DM we see an increase from the previous best score of 86.58% to 86.92%, and 82.71% to 83.27% for PSD.

The increase is likely caused by the disambiguation possible for verbs that are assigned a range of different part of speech tags, such as the verb 'make', which have been assigned this set of tags: 'VB', 'VBZ', 'VBG', 'VBD', 'VBN', 'VBP'. The uniform distribution of frames for each token in PSD might account for the greater increase in f-fscore when using part of speech tags.

Part of Speech Tags Added to the Context Window We will now examine how part of speech tags may affect the accuracy of our model by adding them to the context window. We will run 2 set of experiments: (1) adding part of speech tags to the context window for the token lemma. This is done by concatenating them to the lemma as in our previous experiments. We also run experiments by adding the part of speech tags as separate features. The results are reported in Table 6.10. Similar to the results in Table 6.9, we observe that using part of speech tags as separate features for the context window results in the highest increase in F-score.

The score once we have added part of speech tags are 88.08% for DM and 84.02% for PSD. With part of speech tags we have achieved an increase in our scores with 1.5 for DM and 0.82 percentage points for PSD. We therefore include these from this point on.

Prefixes and Suffixes We will now consider adding prefixes and suffixes as features. For prefixes and suffixes we add character n-grams where $\{n|n > 2 \wedge n < 5\}$. This ensures that we capture various types of prefixes and suffixes that may be constrained by character length, such as the suffixes 'ing' and 'ed'. It is difficult

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
87.46	86.81	86.71	84.04	84.40	**83.30**
86.46	85.85	85.69	81.66	81.91	80.76
86.23	85.63	85.45	80.79	80.91	79.81
87.67	87.00	**86.92**	84.00	84.43	83.27
86.18	85.54	85.37	80.71	80.87	79.76

Table 6.9: Results for adding part of speech tags: concatenated to form (first), concatenated to lemma (second), concatenated to both form and lemma (third), as a feature on its own (fourth), and concatenated to both form and lemma, form and lemma as features on their own, and part of speech tag as a feature on its own (fifth).

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
87.67	87.00	86.92	84.00	84.43	83.27
88.52	88.27	**88.08**	84.24	85.51	**84.02**
88.51	88.26	88.07	84.25	85.52	**84.02**

Table 6.10: Results for adding part of speech tags to the context window, concatenating to lemma (top), as their own separate features (middle), and a combination of both, where we have lemma as their own features, part of speech as their own features, and part of speech tags concatenated to the lemma (bottom).

estimating the effect of adding such morphological features, as the increase in our score is minimal. In Table 6.11 we observe that using only prefixes has an impact where we see an increase in F-score. The other experiments had a negative impact on the scores.

At this point it is worth mentioning that we may have reached a saturation point in the effects that new layers of features may have. In fact, if we had added part of speech tags, or even prefixes and suffixes, at an earlier stage, we might have concluded that some lexical features did not have an effect on the accuracy. Another set of experiments would be to test each set of features on their own, and then examine their cumulative effect after observing their effects on their own. We decided to run our experiments in sequence by adding features as we ran our experiments as we observed an effect of adding features for the most part.

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
88.69	88.29	**88.20**	84.13	85.53	**83.99**
88.37	88.03	87.89	83.69	85.05	83.50
88.56	88.15	88.06	84.06	85.42	83.89

Table 6.11: Results for adding prefix to the main form (first), suffixes to main form (second), both prefixes and suffixes to main form (third).

6.2.3 Syntactic Features

Syntactic dependencies are a very interesting set of features for frame detection. Ambiguous verbs such as 'be', 'make', 'have', 'take', 'say', have a variable set of syntactic dependencies that may have a relationship with their assigned semantic frames.

Let us examine two sentences from our training set: (1) 'It has no bearing on our work force today' and (2) 'They have to do it'. In sentence (1) the frame for the verb 'has' is 'v:e-i-i', indicating that it is a verb that can take two objects, whereas in sentence (2) the verb has been assigned the frame 'v_qmodal:e-h', which is an indication that it is a modal auxiliary verb to the verb 'do'. The frames are from the DM target representation. In such examples the syntactic dependencies, i.e. their edges and labels, can function as parameters for distinguishing between these two type of frames. If we examine the type of errors that our previous experiments produce, it is these type of ambiguous verbs that account for a high percentage of errors.

For syntactic dependencies we will add the dependencies of the target token. We will add dependents by concatenating lemma and label: <lemma_label>. The main verb of a dependency usually has several dependents, so we concatenate them all to a string as its own feature, sorting them by the dependency label in lexical order.

As an example, for the token 'join' we would have the string: <will_aux: board_obj: Vinken_nsubj>, as a feature. However, concatenating dependency labels for the lemma will produce sparse features. We will also examine the effect of adding the head of our main tokens in the same way as our dependents, with the exception that we always have at most one syntactic head, and we therefore add the head's lemma and label concatenated as its own feature.

We also do a combination of adding head and the dependents. The results of these experiments are seen in Table 6.12. We observe that there is an increase for

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
88.97	88.77	**88.50**	84.23	85.61	**84.00**
88.18	88.04	87.81	84.05	85.42	83.86
88.78	88.68	88.35	83.95	85.46	83.77

Table 6.12: Results for adding <lemma_label> syntactic dependencies: only dependents (top), only the head (middle), and a combination of both (bottom).

all experiments for the DM target representation, but a decrease in accuracy for PSD.

In order to solve the problem of sparse features in our previous experiments, we examine the effectiveness of adding just the dependency labels of the target token, concatenated together, and sorted lexically by the dependency label. We also examine the effects of adding a concatenated version of part of speech tags and the syntactic dependency label: <pos_label>. For each target token we will, in the first case, end up with a string of dependency labels, and for the latter a string of tuples consisting of part of speech tags and dependency labels.

The results of these experiments are found in Table 6.13. We observe that these experiments improve the scores for the DM target representation, and using just the labels gives us the best scores, but even there we observe a minimal decrease in score for PSD.

Syntactic dependencies are not useful features for the PSD target representation at this stage in our experiments. We could test syntactic dependencies on their own, but since using syntactic dependencies adds the complexity of using an additional resource; i.e. a syntactic parser, we decided to run these as the last step in our experiments for our first classifier.

We will now turn to semantic dependencies as features for our second classifier. We will not use the syntactic dependencies examined in this section as features once we examine semantic dependencies. We build upon the previous set of features that we ended up with before this section.

6.2.4 Semantic Features

We will add semantic dependencies in the same way that we added syntactic dependencies. However, we omit the results for adding heads of the target token as we observed the same tendency using semantic dependencies as with syntactic dependencies. There was a slight decrease in our scores when adding heads. We

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
88.98	88.76	88.51	84.21	85.50	83.94
89.30	89.06	**88.86**	84.27	85.48	**83.99**

Table 6.13: Results for adding <pos_label> (top) and <label> (bottom) of syntactic dependencies.

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
93.77	94.03	93.69	84.66	86.50	84.70
94.12	94.35	**94.05**	85.68	87.44	**85.74**

Table 6.14: Results for adding <pos_label> (top) and <label> (bottom) of semantic dependencies.

therefore run two sets of experiments by adding a tuple consisting of the part of speech tag of the dependent, and the label between target token and dependent token. We add these as we did the syntactic dependencies by forming concatenated strings sorted lexically by the dependency label.

In Table 6.14 we observe the results for using semantic dependency parsing as part of our feature space. We observe an increase in the F-score of both target representations, with a noteworthy increase from 88.08% to 94.05% for DM, and from 84.02% to 85.74% for PSD. Particularly for the DM target representation, where we see an increase of 5.97 percentage points, the results seem dramatic.

However, it is important to note that when using semantic dependencies as our features, we are testing on a development data set where we have gold standard annotations, and the correlation between the semantic dependencies and semantic frames are artificial, i.e. a semantic dependency parser would not produce dependencies that would have the same fit for our classification task.

In the next section we will run our classifiers on the test data set provided to us by the SemEval-2015 organizers. In order to resolve the issue of using gold standard semantic dependencies for our classifier, we use the semantic dependencies from the Lisbon and Peking parsing systems. With these results we can present a classifier that can be used as part of existing semantic dependency parsing systems.

The results that have been presented in Table 6.14 can be regarded as an upper bound to the accuracy of our system. Given an increase in the accuracy of seman-

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
89.05	89.15	88.61	85.12	85.96	84.72
79.50	80.38	79.35	77.60	76.05	75.45

Table 6.15: Final results on the test data. The classifier using syntactic dependencies on the in-domain (top) and out-of-domain (bottom) data sets

tic dependency parsing systems, we can also expect an increase in the accuracy of our classifier approaching the results presented in Table 6.14.

6.3 Final Results

Before presenting the final results of our classifiers, we recap the feature sets that we use for training and predication. The final results will be divided into three sections. We first run our classifiers against the test data set provided by the SemEval-2015 organizers. When doing so we train the classifiers on all the training data, which now includes the development data used in our experiments. After presenting the results against the test data, we then use data produced by the Lisbon and Peking parsing systems.

6.3.1 Feature sets

We start with the lexical features used by both classifiers. The target tokens form and lemma are included as features. In addition to this a context window of $n = 3$ is used. For the context window we use the token lemma as features.

For the morphological features we use the part of speech tag of the token word, and the part of speech tags of each lemma in the context window.

For the syntactic and semantic features we use the label of the target tokens dependents. These labels are lexically sorted and concatenated together to form a string, which is then used as an individual feature. What differentiates the two classifiers are the usage of dependencies, where the first use syntactic dependencies, whereas the other uses semantic dependencies.

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
94.07	94.20	93.91	87.09	88.03	86.80
87.59	88.03	87.33	80.88	78.44	77.86

Table 6.16: Final results on the test data. The classifier using semantic dependencies on the in-domain (top) and out-of-domain (bottom) data sets.

6.3.2 Running on Test Set

In Table 6.15 we see the results our first classifier running on the test set. Comparing these results with Table 6.13 we observe that we get the same F-score for the DM target representation as in our development set, and an increase in our scores for the PSD target representation. Our final scores for the first classifier is thus an F-score of **88.61%** on the DM target representation, and an F-score of **84.72%** for PSD.

The accuracy of our classifier decreases substantially when we run it on the out-of-domain data set. In Section 6.3.4 we show that the drop in accuracy is seen in the frame classification accuracy of other systems. We do not go into detail as to the possible reasons, and leave the analysis of out-of-domain data outside the scope of our thesis.

Table 6.16 presents the results of our second classifier which uses semantic dependencies as additional features. We observe that the F-score is comparable to the results achieved on our development set, with a final F-score of **93.91%** for DM, and **86.80%** on PSD. It is worth reminding the reader that we are still relying on gold standard semantic dependencies. As mentioned previously, this will most likely act as an upper bound to the possible score that we can reach for this classifier. We will give the name *Oslo* to this classifier.

We have now presented the final results for the two semantic frame classification systems that is the result the experiments presented in this chapter. We will now turn our attention to the results we can achieve when using our second classifier as an extension of the Lisbon and Peking parsing systems that we presented in Chapter 3.

6.3.3 Extending the Lisbon and Peking Parsing Systems

We will now run our second classifier on the semantic dependencies produced by the Lisbon and Peking parsing systems. We will use the results from the closed

DM			PSD		
Precision	**Recall**	**F-score**	**Precision**	**Recall**	**F-score**
88.67	88.03	87.74	84.65	85.55	84.29
88.13	87.34	**86.96**	85.28	86.02	**84.74**
78.95	77.71	**77.09**	75.04	75.30	74.04
77.83	76.53	76.13	75.88	75.66	**74.51**

Table 6.17: Running our second classifier using semantic dependencies from the Lisbon (top) and Peking (bottom) parsing systems. For each system we have run both on the in-domain and out-of-domain data sets.

track for both parsing systems. This is due to the fact that the Peking system only submitted scores in the closed track, and to make the results as comparable as possible. For information on the various tracks of SemEval-2015 see Chapter 3.

We give each of these results the names Lisbon++ and Peking++, as the results could be viewed as extensions to these two parsing systems, i.e. adding our classifier as an additional step after the semantic dependency parsing of these two systems.

6.3.4 Comparative Perspective

In this section we will compare our results with the results of the submissions of SemEval-2015. We present the scores that we use for our comparisons in Table 6.18. We observe that the scores of using gold standard semantic dependencies for the Oslo parsing system is substantially higher than the other systems. The Lisbon++ and Peking++ systems also outperform all other results that are available on previous work on semantic frame classification.

In comparison to the Praha system, the technical details of which can be read in Hajic and Urešová (2015), we also observe a higher score for both Oslo, Lisbon++ and Peking++. The comparison against the Praha system is not completely valid as the data sets used for training and testing are different, and the Praha system use additional information such as parallel texts and lexicons. However, in order to include work outside the submissions to SemEval-2015 we have included the results as a point of reference. The Praha system also do not have data on the DM target representation.

System	DM	PSD
Oslo*	93.91	86.80
Lisbon++	**86.96**	**84.74**
Peking++	84.65	84.29
Minsk	84.01	84.22
Praha	_	82.93
Riga	79.65	79.32
Turku*	75.45	77.14
Peking	82.25	71.29
Oslo*	87.33	77.86
Peking++	**77.09**	74.04
Lisbon++	76.13	**74.51**
Minsk	72.26	77.15
Turku*	61.48	63.62

Table 6.18: The highest F-scores of the SemEval-2015 submissions for the in-domain (top) and out-of-domain (bottom) data sets on the closed track, and gold track (*) for semantic frames.

6.4 Conclusions

In this chapter we have presented the main research of our thesis. We have performed a set of experiments where we have examined the effects of different features on building a semantic frame classifier. Our results outperform previous results. The results of the Oslo classifier are interesting as an upper bound to the accuracy that our system can potentially achieve. However, since we are using the gold standard semantic dependencies as our test data, the results are based on dependencies that are completely correlated with the semantic frames, and therefore produce artificially high F-scores.

Lisbon++ and Peking++ on the other hand are results that rely on the semantic dependencies of existing parsing systems. The results we achieve rival that of previous work on semantic frame classification, both on the DM and PSD target representation, and also both on the in-domain and out-of-domain data sets.

Chapter 7

Conclusion

In this thesis, we have presented an in-depth contrastive error analysis of a selected set of semantic dependency parsing systems that participated in SemEval-2015. The analysis of Chapter 4 pointed in the direction of semantic frame classification as an interesting case to pursue as its own task. This was due to our hypothesis that the results of previous work showed a possibility of improvement based on the accuracy observed on the 30 most frequent frames.

In Chapter 5 we presented the experimental setup for our classification task. We presented the data used for training, development and testing. We also presented the type of features that we would examine. We showed that our feature design would be based on lexical, morphological, syntactic and semantic features.

The experiments that we ran in Chapter 6 where designed so that we could extensively test the features presented in Chapter 5. We selected the best set of features in an additive manner; testing new features based on the previous set of features that we included. We also tested 4 different machine learning algorithms, and finally landed on Support Vector Machines for our classifier.

With the experimental setup in place we set out to create two distinct classifiers; one that would use syntactic dependencies as features, and one that would use semantic dependencies as features. Both classifiers would share the same underlying lexical and morphological features. The classifier using semantic features would be used as a possible extension to the parsing systems examined in Chapter 4.

With this in place we ran the classifier based on semantic features on the gold standard test data, and on the data produced by the Lisbon and Peking parsing system. The first run produced very accurate results, with a F-score of 93.91% for DM and 86.80% for PSD. Since we used gold standard semantic dependencies during classification, this can be considered an upper bound to the accuracy of our classifier. Using the semantic dependencies produced by Lisbon, we presented the Lisbon++ system, achieving a F-score of 86.96% for DM and 84.74% for

PSD. The same run on the semantic dependencies of the Peking parsing system, resulting in the Peking++ system, we achieved a F-score of 84.65% for DM and 84.29% for PSD. These scores outperform previous results on semantic frame classification.

7.1 Future Work

There are several experiments that we could run in order to push the accuracy of our classifiers further. A step that the we would like to pursue is to examine how *Word Representations in Vector Space* might improve the accuracy of our classifier. Concatenating the vectors of approaches from Mikolov, Chen, Corrado, and Dean (2013): Word2Vec, and Pennington, Socher, and Manning (2014): Glove, with the feature vectors used for our training, would be an interesting approach. A more recent approach by Trask, Michalak, and Liu (2015): Sence2Vec, which was used by the authors for word sense disambiguation, would be particularly interesting for feature enhancement and future endeavours.

References

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, *46*(3), 175–185.

Barzdins, G., Gosko, D., Rituma, L., & Paikens, P. (2014). Using c5.0 and Exhaustive Search for Boosting Frame-Semantic Parsing Accuracy. In *Proceedings of the 9th Language Resources and Evaluation Conference (lrec)* (pp. 4476–4482).

Barzdins, G., Paikens, P., & Gosko, D. (2015). Riga: From FrameNet to Semantic Frames with C6. 0 Rules. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 960–964.

Bohnet, B. (2010). Very High Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics* (pp. 89–97). Beijing, China.

Bohnet, B., & Kuhn, J. (2012). The Best of Both Worlds: a Graph-Based Completion Model for Transition-Based Parsers. In *Proceedings of the 13th conference of the european chapter of the association for computational linguistics* (pp. 77–87).

Bohnet, B., & Nivre, J. (2012). A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-projective Dependency Parsing. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning* (pp. 1455–1465).

Carreras, X. (2007). Experiments with a Higher-Order Projective Dependency Parser. In *Emnlp-conll* (pp. 957–961).

Carroll, J. (2000). Statistical Parsing. *Handbook of Natural Language Processing*, 525–543.

Chang, C.-C., & Lin, C.-J. (2011). Libsvm: a Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*(3), 27.

Choi, J. D., Tetreault, J. R., & Stent, A. (2015). It Depends: Dependency Parser Comparison Using A Web-based Evaluation Tool. In *Acl (1)* (pp. 387–396).

Cinková, S. (2006). From Propbank to EngVallex: Adapting the PropBank-Lexicon to the Valency Theory of the Functional Generative Description. In *Proceedings of the fifth international conference on language resources and evaluation (lrec 2006), genova, italy.*

Du, Y., Zhang, F., Sun, W., & Wan, X. (2014). Peking: Profiling Syntactic Tree Parsing Techniques for Semantic Graph Parsing. In *Proceedings of the 8th in-ternational workshop on semantic evaluation (semeval-2014)* (pp. 459–464).

Du, Y., Zhang, F., Zhang, X., Sun, W., & Wan, X. (2015). Peking: Building Semantic Dependency Graphs with a Hybrid Parser. In *Proceedings of the 9th international workshop on semantic evaluation (semeval 2015)* (pp. 927–931).

Earley, J. (1970). An Efficient Context-free Parsing Algorithm. *Communications of the ACM*, *13*(2), 94–102.

Eisner, J. M. (1996). Three New Probabilistic Models for Dependency Parsing: An Exploration. In (Vol. 1, pp. 340–345).

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., & Lin, C.-J. (2008). Liblinear: A library for large linear classification. *Journal of machine learning research*, *9*(Aug), 1871–1874.

Flickinger, D. (2000). On Building a More Efficient Grammar by Exploiting Types. *Natural Language Engineering*, *6*(01), 15–28.

Gaifman, H. (1965). Dependency Systems and Phrase-Structure Systems. *Information and Control*, *8*(3), 304 - 337.

Hajič, J., Hajičová, E., Panevová, J., Sgall, P., Bojar, O., Cinková, S., ... Žabokrtský, Z. (2012). Announcing Prague Czech-English Cependency Treebank 2.0. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC 2012)* (pp. 3153–3160). İstanbul, Turkey: European Language Resources Association.

Hajic, O. D. E. F. J., & Urešová, M. P. J. Š. Z. (2015). Using parallel texts and lexicons for verbal word sense disambiguation. *Depling 2015*, 82.

Hsu, C.-W., & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, *13*(2), 415–425.

Hudson, R. A. (1990). *English Word Grammar* (Vol. 108). Oxford, UK: Basil Blackwell Oxford.

Joachims, T. (1999). *Making Large Scale svm Learning Practical* (Tech. Rep.). Universitat Dortmund, Informatik, AI-Unit: Universitat Dortmund.

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition* (2nd ed.). Upper Saddle River, NJ, USA: Prentice Hall.

Kanerva, J., Luotolahti, J., & Ginter, F. (2014). Turku: Broad-Coverage Semantic

Parsing With Rich Features. In *Proceedings of the 8th international workshop on semantic evaluation (semeval 2014)* (pp. 678–682).

Kanerva, J., Luotolahti, J., & Ginter, F. (2015). Turku: Semantic Dependency Parsing as a Sequence Classification. *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, 965–969.

Kotsiantis, S. B. (2013). Decision trees: a recent overview. *Artificial Intelligence Review*, *39*(4), 261–283. Retrieved from `http://dx.doi.org/10.1007/s10462-011-9272-4` doi: 10.1007/s10462-011-9272-4

Kruijff, G.-J. M. (2002). *Formal and Computational Aspects of Dependency Grammar: History and Development of DG* (Tech. Rep.). Saarland, Sweden: ESSLLI-2002.

Kübler, S., McDonald, R., & Nivre, J. (2009). Dependency Parsing. *Synthesis Lectures on Human Language Technologies*, *1*(1), 1–127.

Kudo, T., & Matsumoto, Y. (2000). Japanese Dependency Structure Analysis Based on Support Vector Machines. In *Proceedings of the 2000 Joint sigdat Conference on Empirical Cethods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics-Volume 13* (pp. 18–25).

Kuhlmann, M. (2010). *Dependency Structures and Lexicalized Grammars: an Algebraic Approach* (Vol. 6270). Berlin Heidelberg, Germany: Springer.

Marcus, M., Santorino, B., & Marcinkiewicz, M. A. (1993). *Building A Large Annotated Corpus of English: The Penn Treebank* (Tech. Rep.). Philadelphia, PA, USA: University of Philadelphia.

Martins, A. F., & Almeida, M. S. (2014). Priberam: A Turbo Semantic Parser with Second Order Features. In *Proceedings of the 8th international workshop on semantic evaluation (semeval 2014)* (pp. 471–476).

McDonald, R., & Nivre, J. (2007). Characterizing the Errors of Data-Driven Dependency Parsing Models. In *Emnlp-conll* (pp. 122–131). Prague, Czech Republic.

McDonald, R., & Nivre, J. (2011). Analyzing and Integrating Dependency Parsers. *Computational Linguistics*, *37*(1), 197–230.

Mel'čuk, I. A. (1988). *Dependency Syntax: Theory and Practice*. Albany, USA: State University of New York Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*.

Miyao, Y., Oepen, S., & Zeman, D. (2014). In-House. An Ensemble of Pre-Existing Off-the-Shelf Parsers. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)* (pp. 335–340). Dublin, Ireland.

Nikula, H. (1986). *Dependensgrammatik*. Malmö, Sweden: Liber.

Nivre, J. (2005a). *Dependency Grammar and Dependency Parsing* (MSI Report No. 05133). Växjö, Sweden: Växjö University.

Nivre, J. (2005b). *Inductive Dependency Parsing of Natural Language Text* (Unpublished doctoral dissertation). School of Mathematics and Systems Engineering, Växjö University.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryiğit, G., Kübler, S., ... Marsi, E. (2007). MaltParser: A Language-Independent System for Data-Driven Dependency Parsing. *Natural Language Engineering*, *13*(2), 95–135.

Oepen, S., Kuhlmann, M., Miyao, Y., Zeman, D., Cinková, S., Flickinger, D., ... Urešová, Z. (2015). Semeval 2015 Task 18: Broad-coverage Semantic Dependency Parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)* (pp. 915–926).

Oepen, S., Kuhlmann, M., Miyao, Y., Zeman, D., Cinková, S., Flickinger, D., ... Urešová, Z. (2016). Towards comparability of linguistic graph banks for semantic parsing. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (lrec)* (pp. 3991–3995).

Oepen, S., Kuhlmann, M., Miyao, Y., Zeman, D., Flickinger, D., Hajič, J., ... Zhang, Y. (2014). Semeval 2014 Task 8: Broad-coverage Semantic Dependency Parsing. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)* (pp. 63–72). Dublin, Ireland.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global vectors for word representation. In *Emnlp* (Vol. 14, pp. 1532–1543).

Sagae, K., & Tsujii, J. (2008). Shift-reduce Dependency dag Parsing. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1* (pp. 753–760).

Sgall, P., Hajičovà, E., & Panevovà, J. (1986). *The Meaning of the Sentence in its Semantic and Pragmatic Aspects* (M. L. Jacob, Ed.). Dordrecht, Holland: D. Reidel Publishing Company.

Taylor, A., Marcus, M., & Santorini, B. (2003). *The Penn Treebank: an Overview.* Springer.

Tesnière, L. (2015 [1959]). *Elements of Structural Syntax*. Klaprozenweg 75G, 1033 NN Amsterdam, Nederland: John Benjamins Publishing Company.

Titov, I., Henderson, J., Merlo, P., & Musillo, G. (2009). Online Graph Planarisation for Synchronous Parsing of Semantic and Syntactic Dependencies. In *Ijcai* (pp. 1562–1567).

Trask, A., Michalak, P., & Liu, J. (2015). sense2vec-a fast and accurate method for word sense disambiguation in neural word embeddings. *arXiv preprint arXiv:1511.06388*.

Yamada, H., & Matsumoto, Y. (2003). Statistical Dependency Analysis with Support Vector Machines. In *Proceedings of iwpt* (Vol. 3, pp. 195–206).

Younger, D. H. (1967). Recognition and Parsing of Context-free Languages in Time n^3. *Information and control*, *10*(2), 189–208.

Zwicky, A. M. (1985). Heads. *Journal of linguistics*, *21*(01), 1–29.