# Boosting for Model Selection in Syntactic Parsing

Rachel Bawden

University

Paris Diderot | Paris 7

Laboratory

ALPAGE – INRIA

**Master's Thesis in Computational Linguistics**

JUNE 2015

# Boosting for Model Selection in Syntactic Parsing

*By:*
Rachel BAWDEN

*Supervisor:*
Benoît CRABBÉ

June 19, 2015

# ABSTRACT

In this work we present our approach to model selection for statistical parsing via boosting. The method is used to target the inefficiency of current feature selection methods, in that it allows a constant feature selection time at each iteration rather than the increasing selection time of current standard forward wrapper methods. With the aim of performing feature selection on very high dimensional data, in particular for parsing morphologically rich languages, we test the approach, which uses the multiclass AdaBoost algorithm SAMME (Zhu *et al.*, 2006), on French data from the French Treebank, using a multilingual discriminative constituency parser (Crabbé, 2014). Current results show that the method is indeed far more efficient than a naïve method, and the performance of the models produced is promising, with F-scores comparable to carefully selected manual models. We provide some perspectives to improve on these performances in future work.

# CONTENTS

# INTRODUCTION

Syntactic parsing is one of the key tasks in Natural Language Processing (henceforth NLP). Assigning syntactic structure to sentences is seen as important for many NLP applications in establishing relationships between words in view to uncovering the meaning behind these relationships.

However up until very recently, the majority of the research in parsing has focused on English, a configurational language, with little morphology and whose word order is very fixed. More recent work has shown a greater interest in parsing typologically varied languages, including those that are morphologically rich, spurred on by initiatives such as the SPMRL (Statistical Parsing of Morphologically Rich Languages) shared tasks (Seddah *et al.*, 2013, 2014). Parsing techniques must therefore be adapted to the fact that the majority of these languages show less fixed word order than English, and also have richer morphology, which could provide key clues as to how syntactic structure should be assigned.

This change in focus coincides with a change in approach to syntactic parsing, provoked by a plateau in the performances of traditional probabilistic generative models. These methods, which produce reasonably high scores for English, due to the fact that word order is so important, have a limited capacity to integrate context and lexical information. Refining these methods results in data sparseness issues and a necessity to perform complex calculations. Many modern parsers therefore rely on discriminative methods and the use of a classifier to score derivations (Hall *et al.*, 2007; McDonald *et al.*, 2005). Context and lexical information can be more easily integrated into the model without facing the same data sparseness issues. The complexity of the problem now lies in a different issue, which is the choice of which features to include in the model.

A model's capacity to predict structure is based on the types of features that are included in the parser. For a model to be expressive, features must be informative, although without being too specific to training data, which can result in overfitting and an inability to make generalisations to unseen data. It is important to select only a subset of possible features to choose the most informative ones and to ensure that parsing is not too time-consuming. It is the task of model selection to select a subset of features to produce an optimal model.

Often the number of possible features is too great to perform a simple enumeration of all feature combinations, which would be the most exact method. Relying on linguistic intuitions, although they can be useful, is a complex and time-consuming way of selecting features due to the interaction of features, and it is not a method guaranteed to produce an optimal result. We therefore focus on automatic methods of feature selection, which rely on heuristic searches of the feature space to produce an optimal model in an efficient way.

The approach we present here is an automatic feature selection method using the machine learning technique, boosting (Freund and Schapire, 1999; Zhu *et al.*, 2006). The method, which has previously been used for feature selection, although not to our knowledge for parsing, has the advantage over standard methods of being far more efficient due to the fact that it fits the model in a forward stagewise additive manner. This efficiency will enable us to explore the feature space more widely than current state-of-the-art methods, which rely on constraining heuristics to make them feasible. Our approach also paves the way for feature selection on more complex data, which may include a greater number of morphological or semantic attributes, an essential step in providing high-performing models for morphologically rich languages.

In this work, we shall focus on the parsing of French, a language which is more morphologically rich than English, but typologically far from being one of the richest. French will serve as a test of our approach in terms of model performance but also of the efficiency of the method. In future work, we hope to also study typologically varied languages, for which annotated data is available, such as Basque, Hebrew, Hungarian and Korean (Seddah *et al.*, 2014).

The structure of the thesis is as follows. We will begin in Chapter 1 by describing the domain of our work, which is the parsing of natural languages, providing a description of formalisms and a state of the art in statistical parsing methods. We also describe in detail the discriminative, phrase structure parser we use throughout this work (Crabbé, 2014) and the way in which features are used to score derivations. In Chapter 2 we provide an overview of model selection methods and discuss the various choices and heuristics available, in particular in the context of parsing. We present our contribution in Chapter 3, in which we describe our approach to feature selection using boosting, before presenting our experiments and results on the French Treebank (FTB) (Abeillé *et al.*, 2003) in Chapter 4. We conclude by reviewing the boosting approach in comparison to a more standard selection method and propose lines of research for further improving the method.

# CHAPTER 1

# PARSING NATURAL LANGUAGES

One of the defining tasks of NLP is syntactic analysis, which seeks to assign syntactic structure to a sequence of tokens or words. The syntactic structure describes the relationship between the words of the sentence and also the way in which they can be combined. The focus in this work is mainly on phrase (or constituency) structure, whereby the syntactic representation produced is a tree structure, with a hierarchical phrasal grouping, as described in Section 1.1. A discussion and comparison of the choice between phrase structure and dependency, the other widely used formalism, is also provided in Section 1.1.

So much attention has been paid to parsing mainly because of the belief of the importance of assigning structure to sentences in being able to access the semantic meaning of sentences, one of the holy grails of NLP. For example, information extraction (IE) relies on being able to analyse the semantic relationship between elements of a sentence to access certain meaningful pieces of information. Within IE, Question Answering involves not only the extraction of information but the formulation of a semantic representation of the question to ascertain what information must be extracted. Many other tasks involving natural language understanding, such as machine translation and text summarisation, would greatly benefit from a full semantic analysis. Syntactic representations are seen as key to obtaining a semantic analysis of the sentence because they impose structure on the sequence of words, which can be exploited to study the semantic relationships between them. A lot of effort has been put into obtaining good quality parsing because, in sequential processing, the quality of semantic parsing is dependent on the quality of the syntactic analysis. Where it is not yet possible to do a full semantic analysis, syntactic analyses can be used as a sort of approximation of the relationships between words and the regularities found in the structure of sentences used for many of these tasks. For example, in machine translation, it is possible to build a transfer system from one language to another based on lexical and syntactic transfer that does not necessary go through a purely semantic phase.

Whether syntactic parsing is used as a stepping stone to semantic parsing or is exploited directly, the aim is to produce as useful and as accurate a syntactic structure as possible. The usefulness of a syntactic representation depends on how the structure is to be used and the accuracy on how well the structures produced correspond to the structure we would have hoped to have obtained. One of the big challenges in developing a parser is ensuring that the parsing is robust, i.e. that it is capable of parsing unseen structures. It must also be capable of adequately disambiguating between multiple possible analyses for a single sequence.

Up until now, much attention has been paid to parsing English, and a further challenge is developing a system capable of parsing typologically different languages, something that has gained interest over the past ten years. There are strong reasons to believe that the syntactic analysis of morphologically rich languages could greatly benefit from the integration of morphological information (such as case or agreement) in parse decisions (Crabbé and Seddah, 2014). An important step in improving parsing of these languages is being able to easily exploit such information if it is made available.

The following chapter will describe and explain the theory and design choices of the multilingual parser used throughout the project (Crabbé, 2014), in particular those that have an influence on the the feature selection methods considered in Chapter 2. The parser is a discriminative phrase structure parser. It uses an incremental shift-reduce algorithm with beam-search and scores derivations using a perceptron classifier, capable of integrating lexical information if provided.

## 1.1 Phrase structure analysis

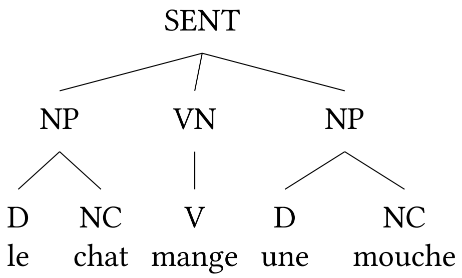
Phrase structure (also known as constituency) analysis is a formalism for representing the structure of a sentence as a hierarchical grouping according to a relation of constituency. Words or sequences of words are organised into phrases (such as noun phrases, verb phrases, prepositional phrases etc.) that define the distribution of the sequences and the way in which they can combine with other phrases to form a sentence.

The sentence “le chat mange une mouche” (en: The cat is eating a fly), shown in Example 1.1 as a sequence of tokens and part of speech (POS) tags, can be represented by the typical phrase structure analysis in Example 1.2[1]. The terminal elements (the leaves of the tree) can be the words of the sentence, their parts of speech, or as shown here, words associated with their parts of speech. An alternative way of representing the same tree structure is by labelled bracketing, as shown in Example 1.3, which is a more compact and more easily machine-readable format often used for treebanks.

(1.1) Le/D chat/NC mange/V une/D mouche/NC

[1] Here the analysis used is that of the French Treebank (FTB).

(1.2)



(1.3) (SENT (NP (D le) (NC chat) ) (VN (V mange) ) (NP (D une) (NC mouche) ) )

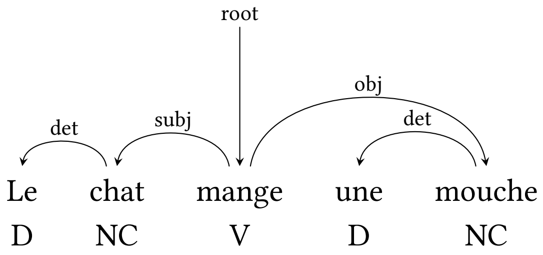
**Constituency and context-free grammars.** Constituency can be modelled with context-free grammars (CFGs), also known as phrase structure grammars, formalised independently by Chomsky (1956) and Backus (1959). A context-free grammar defines a set of rules (or productions) that specify the ways in which phrases can be formed. More formally, a CFG is a quadruplet $G = \langle N, T, S, P \rangle$, where $N$ is a set of non-terminal symbols, $T$ a set of terminal symbols, $S$ the axiom and $P$ the set of productions. Each production is of the form $\alpha \rightarrow \beta$ where $\alpha \in N$ and $\beta \in (N \cup T)*$.

The tree structure in Example 1.2 therefore belongs to the language defined by the context-free grammar productions shown in Example 1.4.

(1.4) SENT → NP VN NP
NP → D NC
VN → V
D → le
D → une
NC → chat
NC → mouche
V → mange

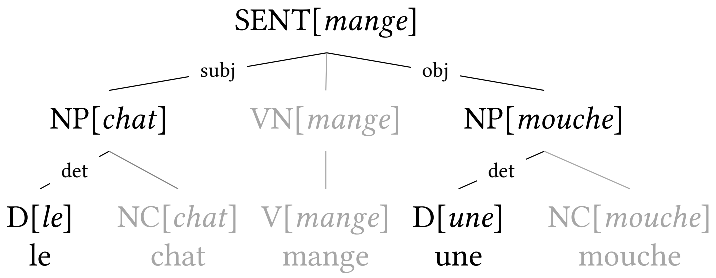
**Dependency Analysis.** An alternative representation of syntactic structure, first formalised by Lucien Tesnière (Tesnière, 1959), is dependency, where relationships between words are represented by binary governor-dependent relations, which are often labelled for a specific function (such as subject, object, modifier, determiner etc.). Example 1.5 shows the same sentence in dependency format, the focus being on the lexical government relationship between a head element and its dependent rather than on phrase-based regrouping.

(1.5)



The two formats, constituency and dependency, are structurally equivalent, provided that only projective dependency trees are considered. Both formalisms associate structure to a sequence of words, and converting between the two formats is possible provided that information is made available to each of the two formats and provided dependency trees are projective. Converting from constituency to dependency requires encoding which branch of a phrase leads to its head, which can be done with a list of head-propagation rules and rules for labelling functional relations. Converting from dependency to constituency requires associating phrase labels to groups containing a governor and its dependents. Therefore whilst structural conversions are possible between the two formats for projective trees, information is required in both directions of translation to provide labelling, highlighting the fact that the two formalisms can be seen to provide different types of information. Example 1.6 shows how a constituency structure is equivalent to a projective dependency tree, provided that functional labels (e.g. subj, obj, det) are provided to label the dependency relations.

(1.6)



One major difference is the fact that in most cases phrase structure trees are necessarily projective, i.e. a constituent contains only contiguous sequences of words, whereas a dependency tree may contain non-projective branching[2]. For this reason, dependency is often said to be more adapted for representing languages with so-called free word order. However in many cases (especially for configurational languages such as English), non-projectivity is a relatively rare phenomenon that can be dealt with post-parsing.

### 1.1.1 Encoding head relations using lexicalisation

One way of integrating information about the head of a phrase into phrase structure is by propagating lexical information associated with phrasal heads throughout the tree in a dependency-like fashion. By providing a phrase structure representation in which the head of each constituent is accessible, we are able to integrate the advantages of using dependency structure into the constituent structure. To do this, there is a need for head-propagation rules, as was the case when converting from constituency to dependency.

Making lexical heads accessible can have advantages for parsing, especially for aiding the disambiguation of parse structures based on the lexical content of the sentence. Syntactic

[2] This is mainly due to the limitations imposed by CFGs. Formalisms with discontinuous constituents do exist, for example linear context-free rewriting systems (LCFRS) (Vijay-Shanker *et al.*, 1987) and there even exist several treebanks for German with hybrid representations, allowing for crossing branches (see Skut *et al.* (1997) for the annotation scheme of the NEGRA and TIGER treebanks).

ambiguity, where more than one syntactic structure is possible for a sentence, is a major issue in parsing. Collins (1999) illustrated the use of head-driven statistical models, adapting methods for probabilistic parsing to lexicalised probabilistic context-free grammars, which resulted in improved performances on the Penn Treebank.

Here, heads are encoded in the grammar via specially marked constituent labels, as shown in Figure 1.

```
                         word            tag            sword
<ROOT-head>
    <SENT-head>
        <NP>
                         Le        DET          Le
                         loyer           NC-head        $UNK$
        </NP>
        <VN-head>
                         sera            V              sera
                         plafonné    VPP-head           $UNK$
        </VN>
        ...
```

Figure 1: An example of the French Treebank in the native parser format with head encoding

We will see in Section 1.4.2.1 how specialised parse actions are used to specify the head-encoding in predicted parse trees and in Section 1.4.3 how lexical information associated with the head of each constituent can be used to score different parse actions.

## 1.2 The parsing of morphologically rich languages

In the early days of statistical parsing, efforts dedicated to improving parsers were centred mainly on English, and in particular the parsing of the Penn Treebank (PTB), which is taken as the reference corpus. The main reason, other than the general anglo-centricity in NLP research, was the lack of good quality training data in other languages. The creation of a treebank is a non-trivial task, which is costly both in terms of time and money. As a result, comparatively very little work had been done on parsing languages other than English. Typologically, English is characterised by the fact that is morphologically poor and word order is quite fixed, which is not the case for the majority of the world's languages. Statistical techniques designed to parse English are therefore not necessarily well adapted to parsing languages for which distinctions are more likely to be lexical than related to word order.

However the tides are changing and more efforts are being focused on the parsing of morphologically rich languages. Methods need to be adapted to be able to incorporate more lexical information than is necessary in English-based models. They need to be able to handle different syntactic structures and more flexible word order and also need to function on a comparatively small amount of data in relation to that of English. Apart from

Petrov *et al.* (2006), whose latent annotations produce high performances on a range of languages, efforts are concentrated on discriminative statistical models (to be described in Section 1.3.2 and the incorporation of lexical information in the form of feature functions. Data is becoming available in a variety of different languages and the creation of SPMRL (Statistical Parsing of Morphologically Rich Languages) workshops and shared tasks has encouraged the development of multilingual parsers. For example, the 2013 SPMRL shared task (Seddah *et al.*, 2013) provided datasets for 9 languages: Arabic, Basque, French, German, Hebrew, Hungarian, Korean, Polish and Swedish, and was aimed at providing more realistic evaluation scenarios. The data used for this shared task is provided in multi-view format, which means that both constituency and dependency structures are supplied for all languages. The reasoning behind providing both formats is that the two structures provide different and useful information, regardless of whether one of the formats may be globally more adapted for a given language. It is generally considered that dependency is more adapted to parsing languages with relatively free word order and constituency to parsing more configurational languages, but in both cases, the other formalism can be useful and we argue that it is important to have both dependency and constituency parsers. It is for this reason that we do not consider one formalism to be superior to the other and here we have chosen to concentrate on the parsing of constituency structures.

## 1.3 Overview of statistical parsing

Parsing strategies can be divided into groups: symbolic parsers and statistical parsers. Before the emergence of statistical parsers in the 1990s, parsers were rule-based and relied on hand-written grammars. Many different frameworks exist, some with highly complex rules to handle subcategorisation, agreement and semantics. Coherent grammars must be developed to handle a wide variety of linguistic phenomena and their advantage lies in their ability to fine-tune the grammar rules. They are often more domain-transferable than statistical approaches because rules are not designed with a specific language style in mind and are instead based on notions of grammaticality, whereas statistical methods are trained on domain-specific data. The disadvantages of these approaches include the specificity of a grammar to a particular language and the time and skill needed to write coherent and robust grammars.

Statistical approaches, which exploit the statistical properties of syntactically annotated treebanks, appeared to provide a relatively simple solution to the problem of having to develop complex and coherent rules by providing a means of extracting them automatically from data. The methods are therefore seen as more transferable from one language to another, provided that data for that language is available. However the availability of language resources is a non-trivial point that should not be overlooked [3]. One of the most important advantages of the data-driven approach and one of the reasons for the interest in developing these methods is that the extracted grammar is a performance grammar, one

[3] Often statistical approaches are said to be less time-consuming than symbolic approaches due to the fact that the latter rely on the development of a hand-written grammar. This is not entirely true given that the time taken to develop the treebank must also be considered and, in the case of a treebank, the complexity lies in the coherency of the annotation protocol.

trained on real data, whilst the hand-written grammar is based on competency and grammaticality. Statistical parsers therefore tend to be more robust in the sense that they can be capable of providing a syntactic structure in all cases. This can be very useful for parsing real-life data in which all syntactic eventualities are difficult to envisage, but these systems usually require a more powerful method of disambiguation because they must allow for many more possible structures. A final advantage of statistical methods is a practical one, in that once the treebank has been developed, making changes to the way in which it is exploited can be far simpler than with symbolic approaches thanks to a separation of data and method.

For these reasons, most modern research in parsing concentrates on statistical methods and the exploitation of annotated data. Here we shall give a brief overview of the two main approaches to statistical parsing: generative modelling and discriminative modelling. We describe the evolution of thought that has led to a renewed interest in more complex modelling and the incorporation of more lexical information.

### 1.3.1 Generative models

Statistical techniques for phrase structure parsing started to become popular in the 1990s, encouraged by the availability of treebanks, such as the PTB which was published for the first time in 1993 (Marcus *et al.*, 1993), and advances in generative statistical methods in the field of speech processing. In parsing, these methods relied on exploiting the structural properties of treebanks to predict the most probable structure for a sentence, and gained interest mainly because of the simplicity of the approach in achieving reasonable performances, at least for English[4]. Inspired by the generative tradition in syntax and by traditional parsing methods such as the Cocke-Kasami-Younger (CKY) algorithm and the Earley Algorithm, the majority of statistical methods relied on probabilistic context-free grammars (PCFGs), which involved modifying the standard algorithms by enriching them with probabilities to disambiguate between competing trees. The simplest version of the generative probabilistic model is one that assigns probabilities to each rule of a context-free grammar extracted from a treebank. These probabilities are usually estimated using the counts in the treebank such that $P(\alpha \rightarrow \beta) = \frac{C(\alpha\rightarrow\beta)}{C(\alpha\rightarrow *)}$, where $*$ represents any right-hand side. The probability of rule $\alpha \rightarrow \beta$ is therefore given by $P(\beta|\alpha)$, with the constraint that $\sum_{\gamma} P(\alpha \rightarrow \gamma) = 1$. A derivational structure can then be assigned a probability by multiplying the individual probabilities of the subtrees (i.e. the CFG rules) of which it is composed.

Since these methods involve assigning probabilities to the individual rules that make up the derivations, they rely on strong independence assumptions; the choice of a constituent label depends only on its direct child nodes, and lexical items are approximated by their POS tags. Therefore, two different sentences which have the same sequence of POS tags will necessarily be assigned the same syntactic structure, regardless of their lexical content. For example, given two sentences “Stig smashes bottles with stones” and “Stig likes bottles

[4] A lot of attention has been (and still is) paid to parsing English and particularly early on in the history of parsing English held a monopoly over most research in NLP.

with labels", each of which could be assigned the same sequence of POS tags "NPP VBZ NN IN NN"[5], it may be preferable to assign the two different structures in Figure 2. However since the two sentences share the same sequence of POS tags, the simple PCFG-based model would assign the same most probable syntactic structure in both cases, and there is therefore no way of making distinctions based on the lexical content of the sentences.

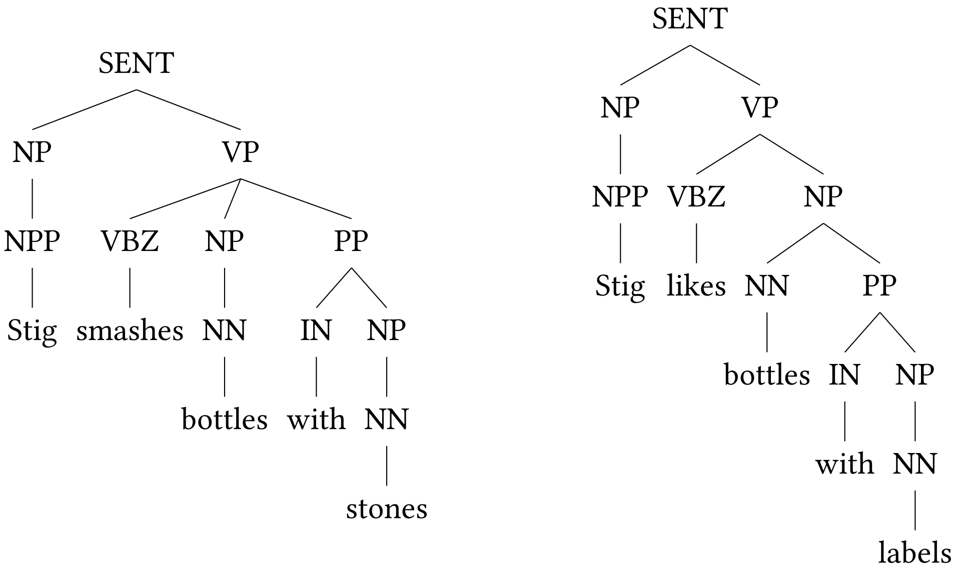


Figure 2: Two different structures for the POS tag sequence "NPP VBZ NN IN NN". Note that these trees would have to be transformed to ensure that they contain the same number of derivational steps in order to compare their probabilities.

The limitations of the generative models therefore lie in their inability to distinguish sufficiently between structures based on their lexical content as well as their inability to take into account non-local contextual information that could help disambiguation.

It was not until the 2000s that these models really took off thanks to new ways of refining the grammars and integrating this information through latent annotations, for example through lexicalised grammars, known as LPCFGs (Collins, 1999; Charniak, 2000) and ancestor and sibling labelling (Johnson, 1998; Klein and Manning, 2003; Petrov *et al.*, 2006). Both methods have the effect of refining POS tags and constituent labels to take into account more information and therefore of weakening independence assumptions.

**Lexicalised Probabilistic Context-Free Grammars (LPCFGs).** An LPCFG is a PCFG in which constituent labels are annotated with their lexical head, which has the effect of splitting constituent labels into finer classes. For example the category VP in Figure 2 can be split into two distinct labels VP[smashes] and VP[likes], making the left-hand side of the VP rules different, as shown in Example 1.7. The probabilities of the structures are no longer necessarily the same, now depending on the specific lexical items in the sentences. The approach has the advantage of conserving the very simple parsing algorithms of PCFGs, but the disadvantage of making parameter estimation more complex and calculations more

[5] The Penn Treebank (PTB) format is used here for tags and for the phrase structure trees in Figure 2.

costly. Refining the labels necessarily makes them sparser, which can lead to very small probabilities for a large number of rules, and this also greatly increases the number of rules in the grammar and therefore the number of possible derivations. Nevertheless,a significant gain in accuracy is seen over simple PCFG-based models.

(1.7) VP[*smashes*] → VBZ[*smashes*] NP[*bottles*] PP[*with*]
VP[*likes*] → VBZ[*likes*] NP[*bottles*]

**Context annotation.** Several other approaches have been developed with the same aim of including contextual information through the decoration of internal tree nodes. The other main branch of category-splitting is through the annotation of contextual elements such as parent annotation (Johnson, 1998; Klein and Manning, 2003), and sibling annotation (Klein and Manning, 2003), enabling non-local contextual information to be taken into account. Yet again, these latent annotations make the models more expressive, but increase the number of different rules and therefore the number of possible derivations that must be calculated. Petrov *et al.* (2006) propose a solution to this problem by providing a system capable of inferring useful category splits, making their models significantly more compact. More recently, a different approach to including lexical and contextual information has emerged, but without having to inject data sparseness into the grammar: Hall *et al.*'s 2014 discriminative parser uses a grammar and surface features related to the span of a constituent, for example span's length, its first and last elements and a span's contextual elements. They show that it is possible to incorporate this information without having to decorate probabilistic trees, going some way to resolve data sparseness issues.

The problem is that pure generative methods, despite the advances made in capturing non-local and lexical distinctions through category splitting, are somewhat limited in their ability to incorporate further lexical information and performances appear to increase very little. This is especially true of parsing typologically varied languages, for which word order is not necessary very fixed. These languages are often also morphologically rich, but this lexical information cannot easily integrated into probability-based methods. With the aim of overcoming the plateau in prediction performances, there is currently a movement to inject complexity into statistical generative models, taking inspiration from the complexity that can be found in symbolic parsers, such as Tree Adjoining Grammar (TAG) parsers, which can easily incorporate lexical information and constraints in feature structures. However current efforts are limited due to the complexity of the formalisms. For example, Kallmeyer and Maier (2010) developed a parser for German using Linear Context-Free Rewriting Systems (LCFRS), which allows for non-continuous constituents and non-projectivity and which is therefore more expressive than context-free systems. The downside is that these more complex systems are currently too computationally expensive to be as high performing as the formalism should allow.

### 1.3.2 Discriminative models

These limitations have led to a recent interest in discriminative, transition-based models for phrase structure parsing (Sagae and Lavie, 2005; Wang *et al.*, 2006; Zhu *et al.*, 2013;

Crabbé, 2014). These methods are inspired by dependency parsing algorithms, for which discriminative parsing is widely used. Whilst constituency formalisms have dominated syntax for English, dependency is often the preferred structure for languages with freer word order due to the facility in modelling non-projectivity and discontinuity. This is one of the reasons why languages such as Russian (the language of one of the leading figures in dependency, Igor Mel'čuk) and Czech (for which there exists the Prague Dependency Treebank) often use dependency rather than constituency. It happens that these languages with a less fixed word order than English also tend to be more morphologically rich, and taking into account lexical information is very important. Discriminative methods, as opposed to generative methods, rely on a classifier to score different elements of the predicted sequence, rather than taking the language produced by a treebank. The use of a classifier to provide scores rather than probabilities simplifies the addition of features, making the technique more transferable from one language to another. What is more, the use of incremental discriminative methods makes parsing very efficient, something that is currently a serious problem for the majority of advanced generative parsing techniques.

For dependency parsing there are two main strategies: transition-based and graph-based. Transition-based parsers construct the structure incrementally by scoring different actions to produce the derivational sequence with the highest score. Inspired by classic parsing methods for computer languages, such as LR parsing, Nivre and Scholz (2004) (also Nivre (2006)) proposed a deterministic transition-based parsing method, which parses in linear time, a huge improvement on the cubic time of most chart-based generative parsing algorithms such as CKY and Earley. In more recent work, Huang and Sagae (2010) extended Nivre's idea to a non-greedy implementation, by using beam-search and dynamic programming to maintain performance in linear time, but widening the derivational search space.

The other main class of dependency parsing is graph-based parsing, which is designed to focus on performing a more exact search than beam-searched transition approaches. They emphasise the coherency of the entire predicted structure when scoring predicted sequences, rather than proceeding in an incremental way. One of the most well-known graph-based approaches is McDonald *et al.* (2005), which seeks to find the maximum spanning tree in an initially complete directed acylic graph (DAG) structure. The algorithm which runs in $O(n^2)$, can produce non-projective trees and has high performances for highly non-projective languages, although the integration of extra features in scoring functions makes the exact search in the DAG very costly.

Some of the most successful dependency parsing algorithms are those that use a combination of the two strategies. For example Nivre and Mcdonald (2008) achieve this by stacking, Sagae and Lavie (2006) by reparsing a weighted combination of parses, and Bohnet and Kuhn (2012) by use of a graph-based completion model, in which a global graph-based approach is used to re-rank derivations in the beam at each stage of a transition-based approach.

Of these two strategies, the first – transition-based parsing – is the one that has attracted interest in terms of phrase structure parsing, thanks to its efficiency and its ability to incorporate lexical information. First attempts by Sagae and Lavie (2005) were directly inspired by Nivre and Scholz's (2004) algorithm for deterministic labelled dependency parsing for English, for which performances were lower than the state of the art for English, mainly due to the use of a greedy search. Further work such as Wang *et al.*'s (2006) parser for Chinese provide more successful results, and more recent work by Zhang *et al.* (2009), Zhu *et al.* (2013) and Crabbé (2014) (this last reference describing the parser used in the current work) use a beam search to increase the number of derivations explored. More details on incremental discriminative parsing and the incremental parsing algorithm shift-reduce will be given in Section 1.4.

## 1.4 Discriminative phrase structure parsing

Discriminative phrase structure parsers are largely inspired by incremental dependency algorithms. They exploit the fact that lexical information can easily be integrated and the fact that incremental parsing algorithms can be very efficient. Here we will describe the incremental, transition-based algorithm (shift-reduce) we use, as adapted to phrase structure parsing. We lay out the derivational search space in Section 1.4.2.1 and describe the use of approximate search techniques to limit the exponential number of structures produced. Finally, we discuss how a classifier and feature functions can be used to disambiguate between potential derivations.

### 1.4.1 Shift-reduce algorithm

The shift-reduce algorithm is an incremental bottom-up parsing algorithm, first described by Knuth (1965) and designed for parsing computer languages and as such was originally deterministic. Sequences are processed from left to right, incrementally building the structure without having to back up at any point. For parsing natural language, the same principle can be used to construct syntactic structures, which are represented as sequences of actions. The main difference is that natural language is characterised by ambiguity, which must be incorporated into the algorithm. This will be discussed as of Section 1.4.2 and for now we shall assume non-ambiguity in the examples used.

The algorithm is usually described as having a configuration made up of a stack and a queue $\langle S, Q \rangle$, the stack containing elements that have already been processed and the queue elements that have yet to be seen. The initial configuration is one in which all words are in the queue and the stack is empty. At each step of the algorithm, an action is chosen, which alters the stack-queue configuration.

For dependency parsing, Nivre's incremental dependency algorithm used actions specific to dependency, namely 'shift', 'left-arc', 'right-arc' and 'reduce'[6]. The parser we use here is not a dependency parser but a consiteuncy parser, but for constituency parsing, the principal is the same. As in the standard algorithm, it uses two classes of action: 'shift' and 'reduce'. 'Shift' transfers the next element of the queue to the top of the stack, proceeding in

[6] Two variants of the algorithm were developed: arc standard and arc eager, both described in detail in Nivre (2008). Whilst arc standard uses only the actions 'shift', 'left-arc' and 'right-arc', arc eager introduces a supplementary action 'reduce'.

a left-to-right fashion to process the next word in the sentence. ‘Reduce’ takes the topmost elements of the stack and reduces them to create a constituent. The derivation is complete when there are no more elements to shift and the elements on the stack have been reduced to the grammar’s axiom.

To illustrate the shift-reduce algorithm, we represent the sequence of shift-reduce steps necessary to produce the phrase structure analysis of the sentence “Le chat mange une mouche” (en: The cat is eating a fly) in Figure 3. Note that ‘reduce’ actions are specific to a constituent type in order to produce a labelled constituency structure.

In Nivre’s dependency version the number of actions for a sentence of length $n$ is therefore $2n - 1$ and the same can be true of the constituency version as long as the ‘reduce’ action is constrained to reduce only and exactly two stack elements. In Section 1.4.2.2 we will explain how in practice, the parser used here produces sequences of actions of length $3n-1$ due to its capacity to also handle unary rules.

## 1.4.2 Derivational search space

The original shift-reduce parsers were those such as LR parsers, designed to be deterministic, which means that at each step of the algorithm there is only a single possible action, or at least a means of disambiguating actions with a lookahead strategy. Parsing natural language is a completely different scenario in which ambiguity is rife, and for a parser to be robust, i.e. capable of parsing unseen sentences, it must envisage ambiguity at each step of the derivation. This means that for a given sentence there exist a large number of possible derivations, which we refer to here as the derivational space. Here we discuss what determines the size of the derivational space, how we can navigate it and methods used to avoid having to explore all possible derivations.

### 1.4.2.1 Properties of the derivations

At each step of the algorithm, the parser is presented with a choice of an action given the current configuration. The number of choices at each step is dependent on the number of possible actions and also on the number of topmost stack elements that can be reduced. If no constraints are imposed by the grammar on the number of daughters allowed for a governing node, the number of possible reductions is greatly increased; for each of the possible constituent labels, as many different reductions are possible as elements on the stack. A lack of constraints also has the serious disadvantage that possible derivations are not guaranteed to be of the same length, which is problematic when it comes to scoring derivations and comparing them (See Section 1.4.3).

It is therefore important to limit the possibilities of reduction by applying certain constraints to the tree structures that can be produced. If the input is limited to a context free grammar in Chomsky reduced form, in which each non-terminal production has exactly and only two non-terminal elements on its right-hand side (corresponding to subtrees of arity 2), and lexical productions are necessarily unary (see Example 1.8), for $n$ the length

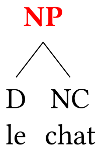
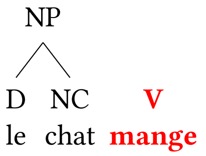
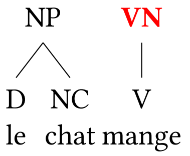
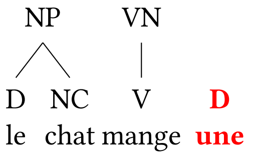
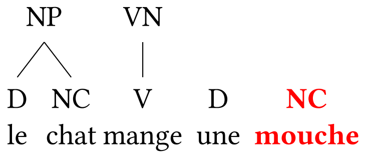
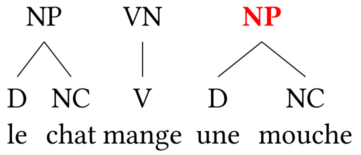
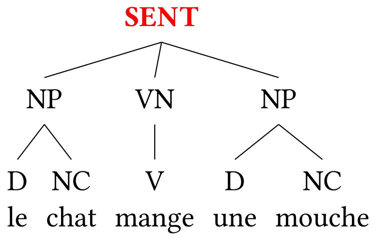
| | **Stack** | **Queue** | **Action** |
|---|---|---|---|
| 1 | | Le$_{D}$ chat$_{NC}$ mange$_{V}$ une$_{D}$ mouche$_{NC}$ | Shift |
| 2 | **D**<br>**le** | chat$_{NC}$ mange$_{V}$ une$_{D}$ mouche$_{NC}$ | Shift |
| 3 | D **NC**<br>le **chat** | mange$_{V}$ une$_{D}$ mouche$_{NC}$ | Reduce$_{NP}$ |
| 4 |  | mange$_{V}$ une$_{D}$ mouche$_{NC}$ | Shift |
| 5 |  | une$_{D}$ mouche$_{NC}$ | Reduce$_{VN}$ |
| 6 |  | une$_{D}$ mouche$_{NC}$ | Shift |
| 7 |  | mouche$_{NC}$ | Shift |
| 8 |  | | Reduce$_{NP}$ |
| 9 |  | | Reduce$_{SENT}$ |
| 10 |  | | END |

Figure 3: The steps of the shift-reduce algorithm for the sentence "Le chat mange une mouche"

of a sentence, the number of possible steps is $2n - 1$.

$$\begin{array}{lcl} A & \rightarrow & B\,C \\ A & \rightarrow & h \\ \multicolumn{3}{l}{\text{where } A, B \text{ and } C \text{ are non-terminal symbols and } h \text{ is a terminal symbol}} \end{array} \tag{1.8}$$

Ensuring these constraints means necessarily transforming the existing input treebank to conform to them, since the majority of treebanks display variable arities. A pre-processing step is therefore responsible for the transformation of the treebank via an order 0 head Markovisation (see Figure 4 and Collins (2003)) to reduce subtrees of arity > 2 to binary trees, followed by a reduction of unary trees (see Figure 5). The transformed treebank is then used to perform the analysis and the resulting trees restored to the original tree format via a post-processing step.

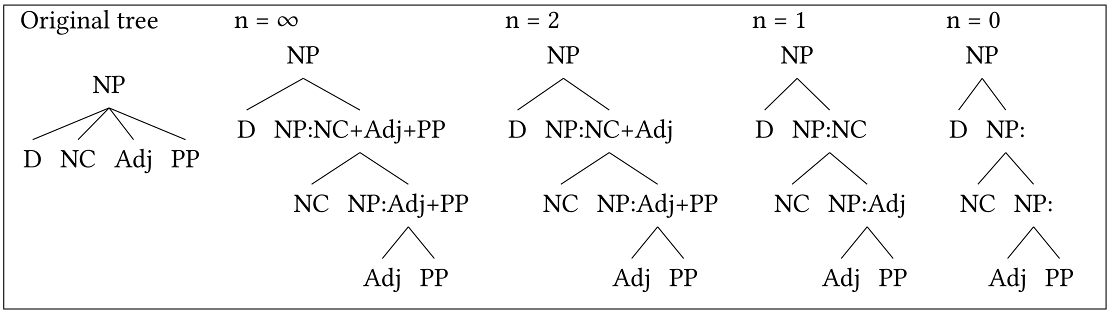


Figure 4: $n$th order horizontal head Markovisation

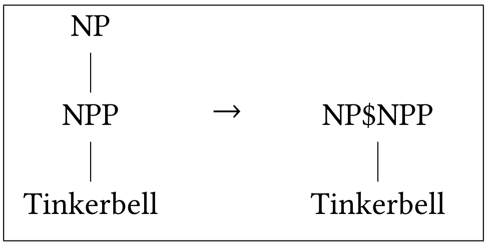


Figure 5: Reduction of unary branches by concatenating node labels

Both pre-processing steps are necessary to ensure that derivations are always of the same length, but they have the effect of increasing the number of possible constituent labels during the algorithm because they introduce temporary symbols, which do not appear in the post-processed analyses. Given that reductions are specific to a constituent label, the number of actions is dramatically increased.

In reality, the parser uses a relaxed version of the Chomsky normal form, in which unary trees are allowed for tag label branching to lexical items. The main reason for allowing these unary branches is that the use of strict Chomsky normal form would have the effect of altering the tag set (as it does with constituent labels), which adds unnecessary sparseness to the data. To ensure that all derivations are of the same length, a further action is introduced, known as 'ghost reduce', which has the effect of adding an action without altering the configuration. With this relaxed version of the grammar, derivations are produced in $3n - 1$ steps.

The different types of parse actions are detailed in Table 1. Apart from 'shift' and 'ghost-reduce', the actions are specific to the constituent type (including temporary constituent labels), which has the effect of multiplying the number of possible parse actions. There are two different types of 'reduce' action in order to propagate lexical heads throughout the tree. 'reduce-left' indicates that the two topmost stack elements are reduced to a constituent and the second element from the top is the head of the new constituent. 'reduce-right', in a similar fashion reduces the two topmost stack elements and indicates that the top element is to be the head of the constituent formed. 'Reduce-unary' is discussed in the previous paragraph.

| | | | |
|---|---|---|---|
| Shift | $\langle S, w_j w_{j+1} \rangle$ | → | $\langle S\ w_j, w_{j+1} \rangle$ |
| Reduce-left X | $\langle \alpha[y]\beta[z], Q \rangle$ | → | $\langle X[y], Q \rangle$ |
| Reduce-right X | $\langle \alpha[y]\beta[z], Q \rangle$ | → | $\langle X[z], Q \rangle$ |
| Reduce-unary X | $\langle \alpha[y]\beta[z], Q \rangle$ | → | $\langle \alpha[y]X[z], Q \rangle$ |
| Ghost-reduce | $\langle \alpha[y], Q \rangle$ | → | $\langle \alpha[y], Q \rangle$ |

Table 1: Illustration of parse actions and the consequences on the configuration. Each item is a pair $\langle S, Q \rangle$ consisting of stack and queue elements.

#### 1.4.2.2 Properties of the search space

As defined in Section 1.4.2.1, derivations are sequences of actions of length $3n - 1$. The search space represents all the possible derivations for a given sequence of wordforms. It is characterised by the fact that at each step of the algorithm, there are $|A|$ possible choices, where $A$ is the number of different actions. The search space therefore increases exponentially at each step of the derivational sequence.

Figure 6 illustrates the exponential space of a simplified shift-reduce algorithm in which there are only two actions and the grammar is in Chomsky normal form, so the length of the possible derivations is $2n - 1$. This reduced space is shown for legibility purposes, but the principle remains the same in our case; as the number of steps in the derivation (directly related to the length of the sentence) increases, the number of possible derivational sequences increases exponentially. The differences between the illustration and our case is that derivational sequences are of length $3n - 1$ and there are a greater number of actions (See Section 1.4.2.1). The number of derivational sequences for a sentence of length $n$ is $|A|^{3n-1}$.

We can see from Figure 6 that the search space can be represented as a tree structure, in which each transition is an action in the sequence. If each of these actions, based on the position in the tree (i.e. the configuration at a certain point in the shift-reduce algorithm), is assigned a score, finding the optimal derivation for a given sentence amounts to finding the maximally weighted path from the start point to the end. A necessary condition is that each of the derivations have the same length for a given sentence, since there is a positive correlation between the score and the number of derivational steps.

Dynamic programming can be used (as in Huang and Sagae (2010)) to reduce the number of nodes at each step as long as there are equivalences between nodes. The principle is to

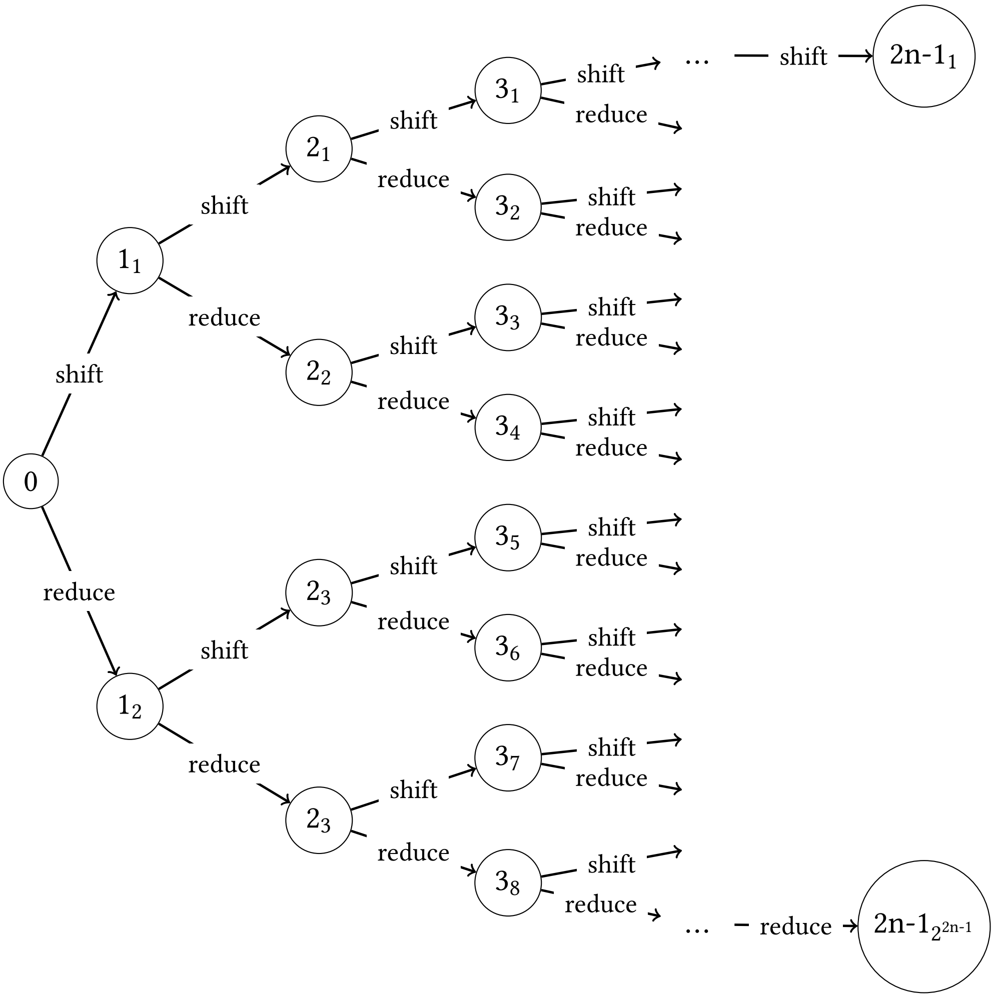


Figure 6: Derivational search space for a shift-reduce algorithm for a grammar in Chomsky normal form for a sentence of length n. At each step of the derivation, the number of possible sequences of actions increases exponentially. The total number of derivational sequences for a sentence of length n is $|A|^{2n-1}$ where A are the possible actions.

regroup overlapping subproblems and to seek an optimal solution through their resolution. Although it is characteristically a technique for reducing the complexity of chart-based phrase structure parsing algorithms, Huang and Sagae succeeded in integrating dynamic programming into an incremental algorithm. We will see in Section 1.4.3.1 how feature functions are used to characterise a particular configuration. Although each configuration is unique to a step of a particular derivation, not all elements of the configuration will be used in the scoring function (this is a choice to be discussed in Chapter 2) and therefore the scoring function can be the same for several nodes in a same step. If this is the case, the nodes that share feature function values within a same derivational step can be regrouped, since the scores of outgoing actions will be the same in all cases. Take for example the sentence "Sassoon runs far in the distance", for which we could expect an ambiguity in the attachment of the adverb "far", either as shown in Example 1.9 or as in Example 1.10.

(1.9) (VP runs (far in the distance))

(1.10) (VP (runs far) (in the distance))

Imagine that the only element of the configuration used to score actions is the first element of the queue. As illustrated in Figure 7, after having shifted the verb “runs” onto the stack, we are faced with two options, to either shift the adverb “far” onto the queue, as would be the case to produce the structure in Example 1.10, or reduce the last stack element to a VP, as would be the case for Example 1.9. If the next action for Example 1.10 is a shift and the next action for Example 1.9 is a reduce, each of the resulting configurations is one in which the first element of the queue is the word “in”. Since the first element of the queue is the only element used to score the following actions from these two points, these two nodes can be considered equivalent and therefore merged (as shown to the right of Figure 7).

In the case where dynamic processing is used, the tree-structured search space becomes a DAG structure, in which the number of internal nodes is inferior or equal to the number of nodes in the tree-structured search space. The same principal of finding the maximally score path in the DAG holds.

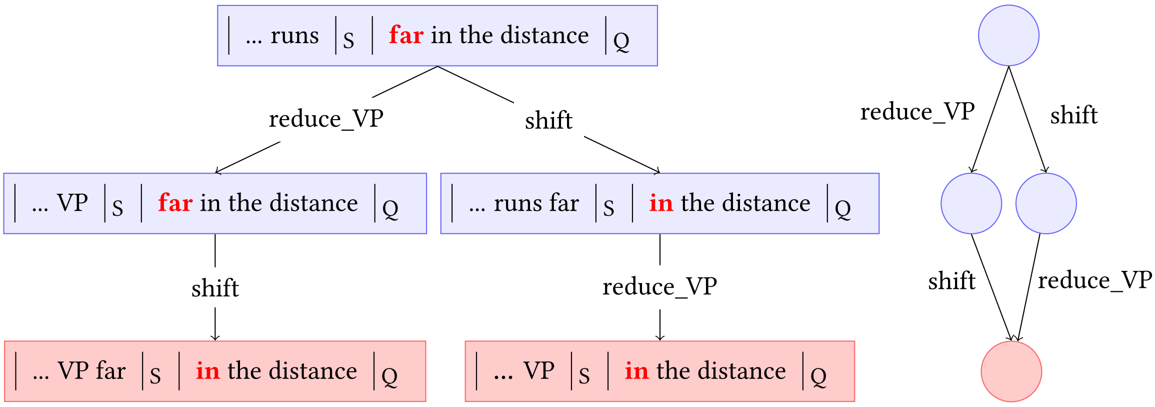

| ... runs |S | **far** in the distance |Q
reduce_VP
shift
| ... VP |S | **far** in the distance |Q
| ... runs far |S | **in** the distance |Q
shift
reduce_VP
| ... VP far |S | **in** the distance |Q
| ... VP |S | **in** the distance |Q
reduce_VP
shift
shift
reduce_VP

Figure 7: An example of the regrouping of derivational steps where the scores of their outgoing actions are guaranteed to be the same. We assume here that the only element used to score different actions is the next element of the queue, highlighted in bold red, which explains the equivalence between the two pink nodes.

#### 1.4.2.3 Approximate search

As shown in Section 1.4.2.2, the derivational search space is exponential in size, and this is a problem for parsing, which must be able to find the best of these derivational sequences for a given sentence. Exploring such a large space, in particular for longer sentences, provides challenges both in terms of memory and time costs.

A common method used, in particular in incremental parsing, is the beam search, which limits the number of sequences kept at each stage of the derivation, retaining the *k* best-scored sequences at each step.

This means that the number of derivations explored is far more limited and avoids the time

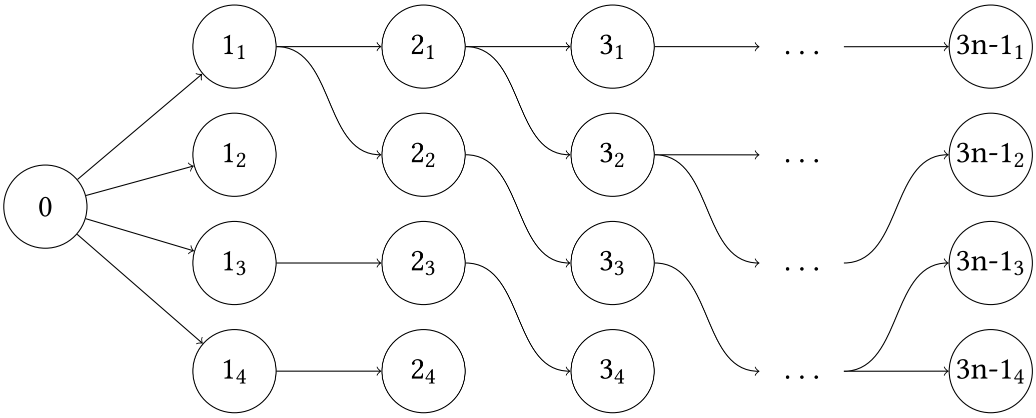


Figure 8: An illustration of the beam search with a beam size of four. At each step of the derivation, only the four best paths are retained.

and memory costs associated with an exponential search. The search is only approximate because not all paths are explored, and the optimally best path might be filtered out before the end of the derivation if its score early on is less than the k-best partial derivations at a given point. If the beam is of size 1, the algorithm amounts to greedily and locally taking the best-weighted transition at each step, as in the first incremental parsing algorithms for natural language (Nivre and Scholz, 2004; Sagae and Lavie, 2005). An infinite beam is equivalent to exploring the entire search space and is usually avoided at all costs. The aim is to reach a compromise between obtaining accurate results and reducing training and prediction times.

### 1.4.3 Disambiguating between derivations

In light of the many different derivations possible for a single sentence, it is necessary to provide a means of ranking them according to how close they are to the desired syntactic analysis[7]. Derivations are scored by using a linear classifier to assign scores to the individual parse actions that make up the derivation and then by summing over these scores.

The classifier used in this work is an averaged multiclass perceptron (described in more detail in Section 1.4.3.2). It is used to assign scores to individual actions based on a certain configuration in the shift-reduce algorithm.

**Structured prediction.** A standard linear classifier would take a set of examples made up of configuration-actions pairs and learn weights to optimise the correct prediction of the actions from each configuration. For a given configuration, an action could then be assigned a score by the classifier. However in the case of parsing, it is preferable for the classifier to learn to optimise not individual configuration-action pairs but the sequence

[7] This can be estimated by the gold standard examples provided in training data.

of actions that make up the derivation (i.e. the sequence of configuration-action pairs), because individual local choices are not necessarily the most coherent for learning the best sequence of actions. The challenge in structured prediction is to optimise a classifier with the aim of producing the best overall derivational sequences and not individual actions. A global optimisation is produced by basing the loss function to be minimised on the entire derivational sequence for a given sentence. Regardless of whether the classifier is optimised locally or globally, the parser must be integrated into the classification process in order to produce the configuration-action pairs to be scored, and in terms of the global model must be used to produce sequences of configuration-action pairs that will be assigned a global score.

An action is scored by carrying out a linear combination of a learned weight vector $w \in \mathbb{R}^D$ and the result of the application of the function $\Phi$ to the configuration $C_i$ and action $a_i$ at a certain point $i$ in the derivation sequence, as shown in Equation 1.11.

$$W(a_i, C_i) = \mathbf{w} \cdot \mathbf{\Phi}(a_i, C_i) \tag{1.11}$$

The score for a derivation sequence of length $M$ $C_{0 \Rightarrow M}$, i.e. $C_0 \overset{a_0}{\Rightarrow} \ldots \overset{a_{M-1}}{\Rightarrow} C_M$ is therefore the sum of the scores of the configuration-action pairs that characterise the derivation, as shown in Equation 1.11.

$$W(C_{0 \Rightarrow M}) = \mathbf{w} \cdot \mathbf{\Phi}_g(C_{0 \Rightarrow M}) = \sum_{i=0}^{M-1} \mathbf{w} \cdot \mathbf{\Phi}(a_i, C_i) \tag{1.12}$$

Since derivational sequences are scored incrementally and the score of a derivation is the sum of the scores of its actions, it is extremely important for competing derivations to be of the same length. The use of a beam search means that at each step of the derivation, the partial scores of each sequence are compared in order to retain the k-best and to prune the rest. The final predicted sequence is therefore the one that is assigned the highest final score and that did not fall off the beam in the process of the derivation.

In the following two sections, we will describe how the configuration is used to produce feature functions (the result of the function $\Phi$) and how the perceptron classifier learns the values of the weight vector $w$.

#### 1.4.3.1 Feature functions

$\Phi(a_i, C_i)$ is a D-dimensional vector of feature functions, each of which is of the form

$$\Phi_j(a_i, C_i) = \begin{cases} 1 & \text{if } a_i = \alpha \text{ and if } C_i \text{ fulfills the condition } x \\ 0 & \text{otherwise} \end{cases} \tag{1.13}$$

where $\alpha$ is a specific action and x is a condition instantiating a value of a particular element of the current configuration. See for example the two features functions in Examples 1.14 and 1.15, the first of which contains a single condition and the second two conditions. In theory they could contain a limitless number of conditions other than the action condition, which is always implicit to the feature. However a maximum of three (as in Huang and Sagae (2010)) is imposed here to reduce the number of possible features.

$$\Phi_{10}(a_i, C_i) = \begin{cases} 1 & \text{if } a_i = \text{shift} \\ & \text{and if “1st queue element word = pot”} \\ 0 & \text{otherwise} \end{cases} \tag{1.14}$$

$$\Phi_{27}(a_i, C_i) = \begin{cases} 1 & \text{if } a_i = \text{shift} \\ & \text{and if “2nd stack element phrase type = NP”} \\ & \text{and if “1st queue element gender = fem”} \\ 0 & \text{otherwise} \end{cases} \tag{1.15}$$

The types of constituents and lexical values available to be used in feature functions are those present in the training data. The configuration is the stack-queue configuration of the shift-reduce algorithm as described in Section 1.4.1 and although in theory the features could instantiate values of any stack or queue element regardless of the size of the stack and queue, in practice, parsers limit the number of stack and queue elements available. As in Huang and Sagae (2010), we limit the number of stack elements for use in the feature functions to three and we also limit the number of queue elements to four, as shown in Table 9. This reduces the number of feature functions, and therefore the number of operations performed to weight each action and, as long as the number is not too constraining, does not have a huge influence on the performance; distant elements tend to be less reliant predictors of the next action than local elements, such as the top element of the stack or the next element in the queue.

Queue items are lexical items and stack elements are partial tree structures in which the leaves are lexical items, and the direct descendants of the first two stack elements are accessible. As described in Section 1.1.1, heads are encoded in the phrase structure analysis, meaning that any constituent is also annotated for its lexical head. This provides another source of values that can be instantiated in feature functions. These lexical items, whether they are in the stack or the queue, are associated with a certain number of pieces of lexical information, obligatorily the wordform and the part-of-speech but also other lexical information as provided in the annotated data, such as gender, number, case, tense etc.

A summary of these configurational elements is presented in Figure 9.

#### 1.4.3.2 Perceptron learning

Weights for individual features are learnt using an averaged multiclass perceptron (see Collins (2002)). The perceptron is an algorithm for learning a classifier from a set of feature functions and the vector $w$, a set of real valued weights. The weights are learnt in an online fashion by iteratively updating the weight vector whenever an example is misclassified. The algorithm is repeated on the set of training data until there is convergence (see Algorithm 1).

In the structured case, the training examples are pairs $(x_i, y_i)$ where $x_i$ is a sequence of words and $y_i$ is the gold derivational sequence. The algorithm is equipped with a func-

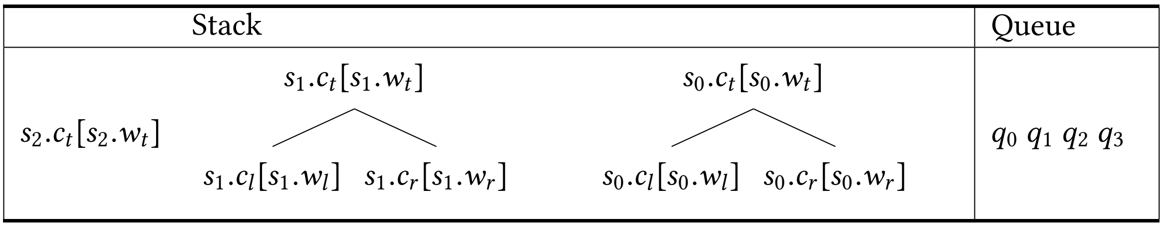


Figure 9: $s_i$ represents the ith element of the stack and $q_j$ the jth element of the queue. Given that the direct daughters of the first two stack elements are also available, $c_t$, $c_l$ and $c_R$ represent the non-terminal of a particular stack element at the three different positions (top, left and right). At each stack element, lexical information about the lexical head is also available, indicated in squared brackets as $s_i.w_{\{t,\,l,\,r\}}$.

tion $GEN(x_i)$ which generates the possible derivational sequences for a given sequence of words. As seen above (Section 1.4.2.3), the use of an approximate search means that only a fraction of all possible derivations will be scored, which is why the algorithm does not use an argmax over all possible sequences. Note that the scoring of a derivational sequence is a linear combination of the weight vector to be learned and a feature representation of the sequence as defined in Section 1.4.3).

**Algorithm 1** Structured perceptron (Collins, 2002)

```
Data = $\{(x_1, y_1), \ldots, (x_D, y_D)\}$
                    ▷ Each $x_i$ is a sequence of words and $y_i$ a derivational sequence $C_{0 \Rightarrow M}$
Initialise $w \leftarrow NULL$
for t from 1 to $T$ do
    for d from 1 to $D$ do
        $z_d \leftarrow \mathrm{argmax}_{C_{0\Rightarrow M} \in GEN(x_d)} \, w \cdot \Phi_g(C_{0\Rightarrow M})$
        if $z_d \neq y_d$ then
            $w \leftarrow w + \Phi_g(y_d) - \Phi_g(z_d)$
        end if
```

**Convergence.** The algorithm is run a certain number of times $T$, but it comes with guarantees of convergence, provided that the data is separable and all updates are valid updates (i.e. they are only made when the predicted sequence is false). We are rarely able to separate natural language data and so whilst convergence to a loss of zero cannot be guaranteed by the perceptron, we can expect there to be convergence to a residual loss (and with residual variation) if the perceptron is run a sufficient number of times. There are methods, such as the averaged perceptron (used here), to counter this problem of convergence. In the averaged perceptron, rather than the weight vector of the final iteration being used, the average weight vector over all iterations (a total of $D{\cdot}T$) is calculated, therefore giving more weight to more consistent weight vectors and avoiding a bias towards the later examples. The moment at which the perceptron converges is linked to the number of updates made, the number of examples and the size of the model (i.e. the number of feature functions used to score the derivations); in general, the larger the model, the longer the perceptron takes to converge. Furthermore, the more feature functions there are in a model, the more calculations are necessary to predict a sequence, and so the longer the training and parse times.

**Perceptron and beam search.** It may appear strange to consider that an update may be invalid, since the algorithm specifies that an update should only occur when the predicted output does not match the gold output. However the use of inexact searches such as a beam search means that valid updates are not always guaranteed. As evoked in Section 1.4.2.3, the correct derivation may be pruned before the final step of the derivation if its partial derivation scores lower than the k-best partial derivations up to this point. Therefore, even if the derivation would have scored highly enough in its final steps to be the best scored overall, the pruning procedure of the beam search means that it falls off the beam and is no longer considered. This is illustrated graphically in Figure 10, in which the beam search is crudely represented by the shaded area on the graph. At step 5, derivation C falls off the beam, since its score is lower than the k-best partial derivations. Had the derivation been continued for C and its score calculated until the final step 8, its final score would have been higher than the best-scored derivation according to the beam search, in this case derivation B. An update performed here would be invalid, due to the fact that the incorrect sequence was chosen not because of poorly optimised feature functions but because of the use of an approximate search.

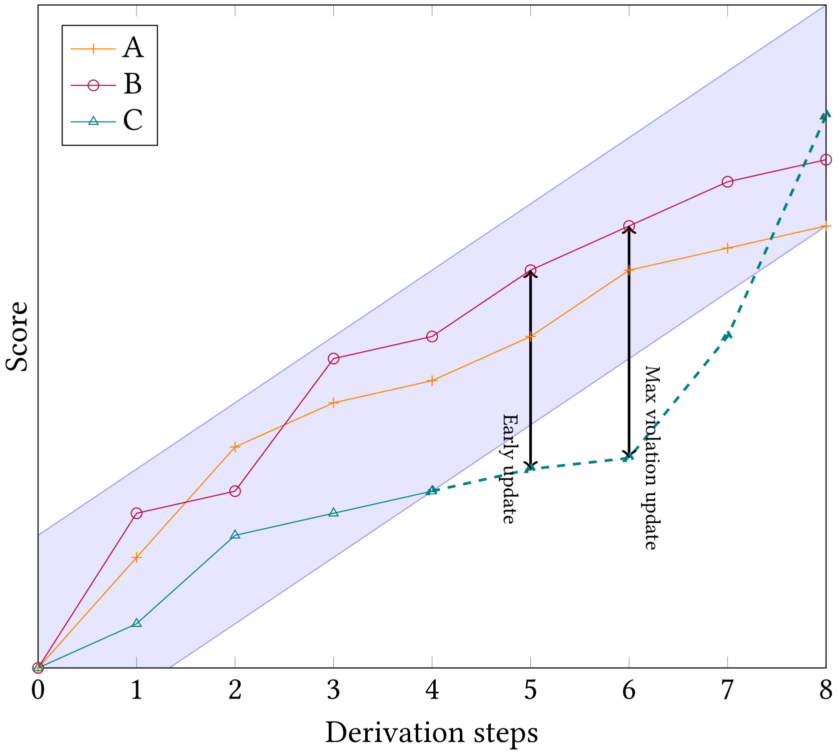


Figure 10: An illustration of the growing scores for three derivational sequences, where derivation C would have been the highest scoring but falls off the beam at step 5.

To ensure valid updates, even when using an inexact search, different strategies can be used. The first to be developed was ‘early updates’ by Collins and Roark (2004). It consists of performing the update of the weight vector at the point in the derivation where the correct derivation falls off the beam, only updating on the partial derivations up to this point. In Figure 10, this corresponds to a partial update at step 5. The disadvantage of this approach is that updates are only performed on partial derivations and more iterations are often needed for the perceptron to converge. An alternative solution, 'max violation update', was provided by Huang *et al.* (2012), whereby the update is performed at the point in the derivation where there is a maximum violation, i.e. where the difference between the best-scored derivation and the correct derivation is greatest, corresponding to step 6 in the illustration.

# CHAPTER 2

# MODEL SELECTION

Model selection is the task of choosing which feature functions are to be used to score derivations. This is a crucial part of designing a parser, because features define the capacity of the model to be expressive and capture generalisations between the input and the structured output. Too few features or the inclusion of only non-informative ones may result in an insufficient ability to capture the relationship between configurations and actions. Too many features could result in the correlations being too specific to the training data and therefore not generalisable, which is known as overfitting. Furthermore, the time taken for parsing is directly linked to the number of calculations that need to be performed. The higher the dimensionality of the model, the longer the parse times will be. In discriminative models, which can allow for a very large number of dimensions, reducing the number of dimensions in the model is crucial.

The aim of model selection is to select an optimal model from a set of possible models. In Section 2.1 we will see how there are multiple criteria for optimality in the case of parsing, including generalisation performance, compactness and speed of the method used. We will illustrate how this notion is also dependent on the level of granularity at which feature selection is performed, a choice which also defines how many possible models are in the search space. It should become clear from this discussion that enumerating all possible models is an unfeasible task, and in Section 2.2.1 we provide some clues as to the limitations of linguistic intuitions when manually selecting a model, in particular when it comes to the interaction of features. We dedicate the remainder of the chapter to automatic feature selection methods, providing a general review of the literature in Section 2.2, before describing those specific to feature selection in parsing in Section 2.3.

## 2.1 What does it mean to select an optimal model?

### 2.1.1 Defining optimality

Defining what constitutes an optimal model relies on having a set of principles for comparing them. Despite the fact that the topic has been widely studied in the literature, the problem is that there is not a single unique criterion for defining optimality when performing model selection. We identify three different optimality functions for model selection: i) the bias-variance trade-off, ii) model compactness and iii) the speed of the feature selection method.

Model selection is often linked to the bias-variance trade-off (Geman *et al.*, 1992); a compromise between choosing a model that can represent the training data well (shown by low training error) and one that can generalise well to new data (shown by low generalisation error). The bias is a measure of the degree to which a model characterises the training data, calculated as the difference between the predicted values and the true values. A model which fits the training data well is therefore characterised by a low bias. The variance measures the specificity of the model to a given training set. It is calculated as the difference between the predictions of the model over different training sets. If a model is very dependent on the specific points of a data set, predictions will vary widely depending on the training data used and variance will be higher. In a good model, we therefore seek to minimise both bias and variance. These two concepts are often conflicting in that low bias (high accuracy on training data) often entails high variance (a model that overfits the training data), and low variance (a very general model) often entails high bias (one that cannot capture regularities in the data and therefore is said to underfit). Finding an optimal model is therefore a trade-off between the reduction of bias and variance. The ability for a model to be generalised to new data is clearly an important factor. Models are often fit using generalisation accuracy as a guide, or cross-validation if the data set is small. Smaller models (with fewer parameters) are often preferred for computational reasons; however expressivity of the model is important for ensuring a sufficient ability to separate data points. Expressivity or complexity of a model can be expressed using the Vapnik-Chervonekis (VC) dimension (Blumer *et al.*, 1989), which is the largest number of points that the model is able to shatter (separate) given all possible combinations of labels of the data points. Although not always directly linked to the number of features in the model, in linear models it is usually the case that a model with more features has a greater VC dimension. This dimension is important because it enables the calculation of the upper bound of the generalisation accuracy of a model. The higher the dimension, the greater the upper bound and the greater the chance the model has of separating test data.

Bearing this in mind, the general strategy to be adopted is one in which generalisation accuracy is the defining criteria for choosing a model over another. In the case of two models quasi-equivalent in generalisation accuracy, the smaller model is preferred, not only following Occam's razor but also in order to optimise training and parsing speed. However, a third criterion, the speed of the feature selection itself, must be taken into account. Given a very large number of features, different strategies must be adopted to limit the time taken to choose the model in the interest of feasibility and practicality. In practice, the choice is therefore a three-way trade-off between generalisation accuracy, compactness of the model and the speed of the feature selection.

**An important remark.** It is important to mention that in statistical parsing there is rarely a unique optimal model for a given problem. One reason for this is the correlation of features. If two features are very highly correlated to each other, one may be redundant if the other is already included in the model. The decision to include one over the other may be arbitrary or a function of the algorithm or the parameters used and not necessarily indicative of their comparative pertinence. For this reason, the interpretation of statistical models to make theoretical judgements should be done with care, bearing in mind this interdependence of model features and the fact that certain results may be an artifact of the chosen method.

### 2.1.2 Model search space

Although we have identified criteria to define optimality in the case of feature selection, the notion remains incomplete without mentioning the set of possible models in which selection takes place (which we refer to as the model search space).

Determining the size of the model search space is important for the following chapter in which we discuss feature selection methods. This is because the feasibility of different feature selection methods is often highly dependent on the number of different models that can be produced and the number of elements in the search space, especially if complex training is incorporated into the feature selection process. However it is also important for the notion of optimality itself; optimality is dependent on the level of granularity of the model selection, i.e. to what extent we make distinctions between models, which is determined by the basic units we choose to manipulate.

#### 2.1.2.1 The basic units of a model

The number of possible models and the granularity of the distinctions between models are defined by the basic units that are manipulated in model selection. The number of models is equal to the number of different combinations of the basic units. So if there are $m$ basic units, there will be $2^m - 1$ possible combinations, if we consider that a model must contain at least one unit.

The smallest basic units used for scoring derivations are features. However larger groups such as feature templates can also be manipulated. Using larger units has the effect of reducing the search space, which increases the efficiency of the feature selection methods, but also reduces the ability to fine-tune models, because fewer distinctions can be made between them. There can be several reasons for choosing a larger basic unit over a smaller, more precise one: readability, gain in speed, and compactness of representation.

**Feature functions.** A model is defined by the feature functions it uses to score derivations. These feature functions define the expressivity of the model and its capacity to make good predictions. They are the elementary blocks for scoring actions and therefore ideally, if other considerations such as search space size and efficiency are disregarded, feature selection should be performed at the feature level. However the feasibility of this fine-grained approach lies in the size of the model search space.

In all statistical models in which features instantiate particular values, the number of features is necessarily dependent on the number of different values for each attribute type used, as defined by those present in training data. The number of combinations is also influenced by the number of different conditions that can be included in a feature function, given that it is possible to create interactions using a logical 'AND'. This factor is fixed to avoid having an infinite number of features. Finally, for a shift-reduce parser, we can add the fact that the number is also dependent on the number of different positions of the shift-reduce configuration that can be used in feature functions.

To give an idea of the number of feature functions in a model, we provide an example using the specific possibilities and constraints of the shift-reduce parser used throughout this work. Let us consider the case where the only lexical information available is the wordform and the POS tag. Let S denote stack items considered[1] and Q queue items considered, A the set of possible actions, Σ the set of all possible wordforms, Γ the tagset and C the set of constituent labels[2].

Each feature is inherently specific to an action, so here I shall consider that a feature containing one condition is a feature with one condition other than the one pertaining to the action. For example, the following feature function contains a single condition:

$$\Phi_1(a,C) = \begin{cases} 1 & \text{if } a = action1 \text{ and if } q0 = word1 \\ 0 & \text{otherwise} \end{cases}$$

The number of different conditions is calculated by multiplying the number of values for each type of lexical information (here this is $|\Sigma| + |\Gamma|$) by the number of possible positions that can have associated lexical information ($|S| + |Q|$) and adding the number of possible non-terminal categories of each of the stack positions:

$$\text{nconditions} = \Big(|\Sigma| + |\Gamma|\Big)\Big(|S| + |Q|\Big) + \Big(|S| \times |C|\Big)$$

The number of possible features with a single condition is the number of possible conditions multiplied by the number of possible actions.

$$\text{F}_1 = |A| \times \Bigg(\Big(|\Sigma| + |\Gamma|\Big)\Big(|S| + |Q|\Big) + \Big(|S| \times |C|\Big)\Bigg)$$

Features can have a maximum of three conditions, and calculating the number of features

[1] Given that the first two stack elements are tree structures rather than simple nodes, here I shall consider the left and right daughters of the first two stack elements to be separate stack elements to help the quantification process. Therefore in a situation where we consider three stack elements (of which the first two are tree structures), the number of stack items S would be 7.

[2] In our model, the stack elements are also associated with a constituent label.

with $n$ conditions amounts to calculating the number of possible combinations of conditions, multiplied by the number of possible actions:

$$\mathrm{F}_n\text{= } |A| \times \binom{nconditions}{n}$$

For example, consider the scenario S (based on the training set for the version of the FTB used for the 2013 SPMRL shared task (Seddah *et al.*, 2013)), which has a wordset of 27,472 words, a tagset of 31 tags, 182 possible actions, 120 possible constituent labels[3], seven stack items (including the two daughters of the first two stack items) and four queue items and a maximum number of three conditions (in addition to the action condition) for each feature function, the total number of features is as follows:

$$\begin{aligned}
F_1 &= 55{,}213{,}886 \\
F_2 &= 8.375 \times 10^{12} \\
F_3 &= 8.469 \times 10^{17} \\
F_{total} &= 8.469 \times 10^{17}
\end{aligned}$$

which makes an impressive total of $2^{8.469 \times 10^{17}}$ possible models.

**Feature templates.** It is common practice in parsing, especially when there are a huge number of features, to manipulate a larger unit than the feature when defining a model, whether this is done manually or automatically. These units, known as feature templates, abstract away from the individual values of features and code simply the type of information used, which acts as a way of grouping features into groups of those using the same elements and information types. This can be particularly important for the readability of the type of features present in a model, and is often the method used when manually defining what types of information from the configuration should be used. By extension, it is also often the method used for automatic selection for the same reasons and also because it is often useful to regroup features into groups to reduce the size of the search space.

For example, the template 'T(1) = q0(word)' regroups all features pertaining to the word of the first element of the queue in the shift-reduce configuration. Since each feature function is specific to an action and to the values of the condition, a template regroups a number of different features. In a simplified model, in which there are only two possible actions $action1$ and $action2$ and two possible word forms $word1$ and $word2$, this template will create 2×2 features, representing all the different combinations of values for q0 and actions:

$$\Phi_1(a, C) = \begin{cases} 1 & \text{if } a = action1 \text{ and if } q0 = word1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Phi_2(a, C) = \begin{cases} 1 & \text{if } a = action1 \text{ and if } q0 = word2 \\ 0 & \text{otherwise} \end{cases}$$

[3] The number of constituent labels is large due to the introduction of temporary symbols in the binarisation process. The number of actions is dependent on the number of constituent labels because the actions reduce_left, reduce_right and reduce_unary are specific to a type of constituent

$$\Phi_3(a,C) = \begin{cases} 1 & \text{if } a = action2 \text{ and if } q0 = word1 \\ 0 & \text{otherwise} \end{cases}$$

$$\Phi_4(a,C) = \begin{cases} 1 & \text{if } a = action2 \text{ and if } q0 = word2 \\ 0 & \text{otherwise} \end{cases}$$

In a more realistic scenario, the number of features generated by a template is much greater. If Σ represents all possible wordforms and A the possible actions, the number of features instantiated by this template is $|A| \times |\Sigma|$, representing all the different combinations of values. For a vocabulary of just 1,000 words, and five actions, this single template produces 5,000 features[4].

Templates are used primarily as a way of specifying features, but this regrouping into classes of features can also be beneficial for defining the feature space. If features are automatically grouped into these classes, manipulating these classes of features rather than individual features is a way of dramatically reducing the search space, and therefore the time taken to run the selection algorithm.

For example, for the scenario S, the total number of templates is 4,088. See Table 2 for a comparison.

| Number of Conditions | Number of Features | Number of Templates |
|---|---|---|
| 1 | $5.521 \times 10^{7}$ | 29 |
| 2 | $8.375 \times 10^{12}$ | 406 |
| 3 | $8.469 \times 10^{17}$ | 3654 |
| All | $8.469 \times 10^{17}$ | 4088 |

Table 2: Comparison of feature numbers and template numbers

This simplification comes at a cost; by performing template selection instead of feature selection, the search is more approximate and the final model likely to be larger, since features are added by batches. There is the risk that a very important feature be missed if it is grouped in a template with less important features, and so the final model is likely to be less good than a feature selection model that selects individual features. However the gain in the speed of the selection is particularly important, especially given that selecting individual features could well be unfeasible depending on the chosen algorithm.

**A note on vocabulary.** Vocabulary is variable in the literature and the term 'feature' does not always have the same value for all authors. In many cases, 'feature' is often used to refer to a feature template (for example in Nilsson and Nugues (2010), Ballesteros and Nivre (2014)), and therefore in the majority of cases it is template selection and not feature selection that is performed. We will follow in this tradition and also use feature templates as our basic units and will make the distinction between the terms 'feature' and

[4] In practice, the number of features is huge, and to avoid memory problems, sparse representations of feature vectors must be used, where only feature functions of value 1 are stored.

'feature template' where necessary. Where we speak of 'feature selection', this should be understood as the task of selecting feature templates in our specific case, and 'feature' will be used in the general case where it is not specified what basic units are to be selected. The term 'feature' is also sometimes used to refer to the different attributes in the data (the different kinds of information provided, such as the wordform, the POS tag etc.). We will refer to these items as attributes or variables.

#### 2.1.2.2 Consequences of the choice of unit

We have already mentioned the fact that the number of possible models is dependent on the number of basic units chosen to make up a model. This is relevant because model selection is the selection of an optimal model amongst all possible models. If the search space is sufficiently small, enumerating all possible models to select the highest performing is a trivial task. However an exhaustive search of the model space is often impossible due to a very high number of possible models. For example, the training of a single shift-reduce parser model can be several hours for a model containing approximately 50 feature templates. The training time is related to the number of calculations necessary (greater for a larger model) and the time taken for the perceptron to converge, and in general the more templates added to the model, the longer this will take. If there are 4088 possible feature templates, and therefore $2^{4088}-1$ possible models, we can easily see that training all models is an unfeasible task. Heuristic methods, such as those mentioned in the following chapter must be used to approximate the search. These heuristic methods often require manipulating the basic units of models, for example by selecting models unit by unit. In this case, although the method does not require enumerating all possible combinations, a search of the basic unit space is nevertheless necessary and the time taken to perform the search is dependent on the number of basic units available.

We shall see that in parsing, methods are often used to reduce this feature space to accelerate the time taken to perform feature selection, either by making the basic units larger or by fixing constraints as to the order in which basic units are searched.

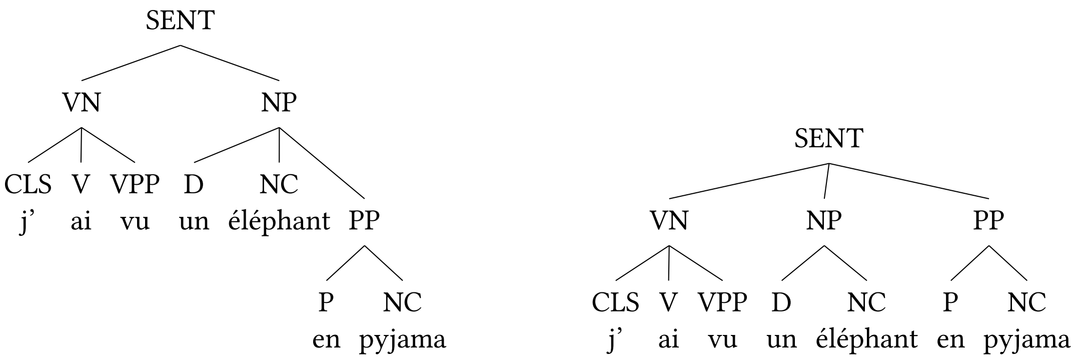
## 2.2 Overview of feature selection methods

We have established in the previous section that enumerating all $2^m - 1$ possible models to select the one with the highest generalisation accuracy is not feasible; even when grouping features into feature templates, the cumulative training time of all models is unreasonably long. The alternative is to find ways of approximating the solution, either manually or automatically.

### 2.2.1 Manual methods

Many statistical parse models rely on hand-chosen feature templates, selected according to linguistic intuitions and a certain amount of trial-and-error, for want of a better method. The resulting models can be high performing but there is no guarantee that the model is optimal, and choosing a model manually can be a very time-consuming task. The selection process can be guided by linguistic intuitions, which can help to target particular values or contextual elements. We can cite for example the well-known problem of the ambiguity of PP-attachments and that of coordination, but we stress that any remarks are given as an example and are not guaranteed to hold true in all cases.

**PP-attachment.** Consider the sentence “J’ai vu un éléphant en pyjama” (en: I saw an elephant in pyjamas). There is a real ambiguity as to the attachment of “en pyjama” either to “éléphant” or to “j’ai vu” to produce the two analyses shown in Figure 11.



(a) The elephant was in pyjamas

(b) I was in pyjamas

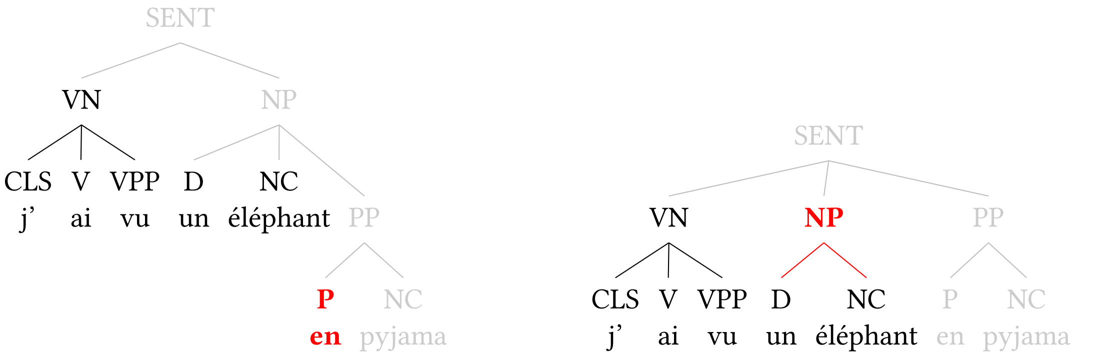
Figure 11: Ambiguity in the attachment of the prepositional phrase

In this case the ambiguity is real, but in many cases the it is possible to disambiguate the structure based on the words of the sentence, and we can apply linguistic intuitions to decide which elements might be useful in aiding the disambiguation. Let us take the incremental shift-reduce algorithm used for our parser. The shift-reduce conflict appears once “éléphant” has been shifted onto the stack, as shown in Figure 12.

| Stack | | | Queue | |
|---|---|---|---|---|
| $s_2$ | $s_1$ | $s_0$ | $q_0$ | $q_1$ |
| VN[***vu***] | D[*un*] | NC[***éléphant***] | **en**/P | **pyjama**/NC |

Figure 12: The configuration just before the shift-reduce conflict, with key elements highlighted.

We can either shift, as in Figure 13a or reduce as in Figure 13b. We can speculate as to which elements could potentially be useful for making the choice. For example, the lexical head at position $s_2$ could be a good trigger in this case, because certain verbs are associated with only one type of PP-attachment. If we were to replace the verb “vu” by “voulu” (en: wanted) to create the similar sentence "J'ai *voulu* un éléphant en pyjama" (en: I wanted an elephant in pyjamas), the preferred reading would be the structural analysis of Figure 11a. Other potentially important elements are the lexical item of the preceding NC (here at position $s_0$) and the lexical item of the noun within the prepositional phrase (here at position $q_1$). Finally, the preposition itself, situated at $q_0$ can also be an important trigger.



(a) Shift the word "en"

(b) reduce the topmost stack elements to NP

Figure 13: Shift-reduce conflict for PP-attachment

However, although certain elements can be identified as key, the configurations are susceptible to variation, which may alter the position of key elements, making them difficult to predict. For example, if instead of a normal elephant, the elephant were blue: "J'ai vu un éléphant bleu en pyjama", there would be one extra element "bleu" on the stack and the other stack elements would be shifted down one place. This means that the key elements are no longer found at the positions $s_2$ and $s_0$, making the identification of key positions particularly difficult, even for isolated phenomena. Many such variations exist, making it extremely difficult to make reliable generalisations.

**Coordination.** The attachment of coordinated elements is another well-known problem of ambiguity in parsing and far more complex to model than PP-attachment. Consider the English sentence "I saw a mouse and an elephant squealed", whose syntactic structure is presented in Figure 14. The structure presents sentential coordination as contrasted with nominal and verbal coordination structures shown in Figures 15b and 15a.

To identify key positions in the shift-reduce configuration that could help choose where to attach the conjunct, we again present the shift-reduce configuration at an intermediary point in the derivation, after having shifted the word "mouse", as shown in Figure 16. The potential candidates for indicating whether to shift the "and" straight away (creating a nominal coordination) or to reduce the two topmost stack elements (to perform either a verbal or sentential coordination) could be the category of $s_1$ and of $s_0$, the lexical form of $q_0$ and the tag of $q_3$. The logic behind this is that the category of $s_1$ and $s_0$ could indicate whether we already have the verbal element of the current sentence. For example, had the category of $s_1$ been an adverb such as "Yesterday" and supposing that $s_2$ is empty, "a mouse" and "an elephant" could form a coordinate subject noun phrase. The lexical form of $q_0$ could be an indication because certain conjuncts may be preferred for certain types of coordination. For example "but" would suggest that nominal coordination is not

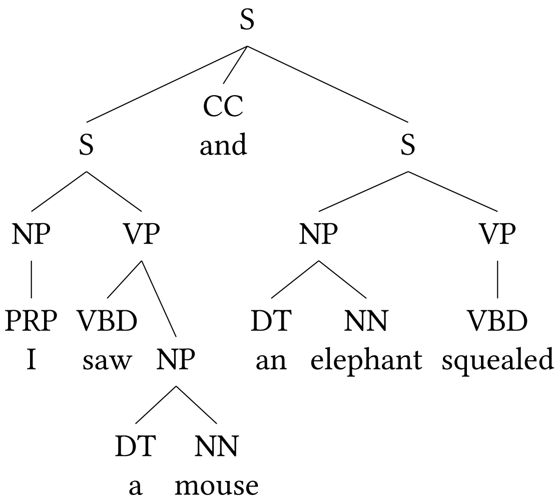
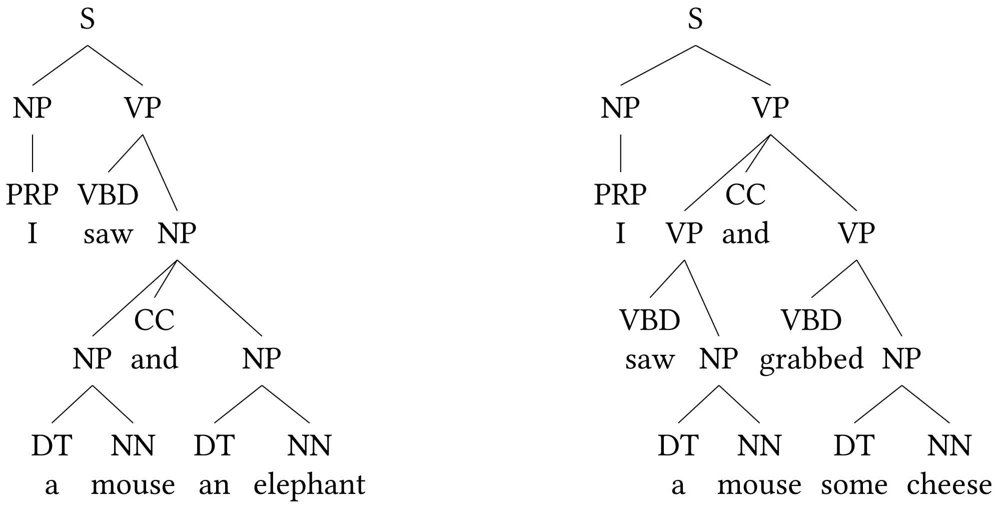


Figure 14: Sentential coordination construction



(a) Nominal coordination construction

(b) Verbal coordination construction

Figure 15: Coordination constructions

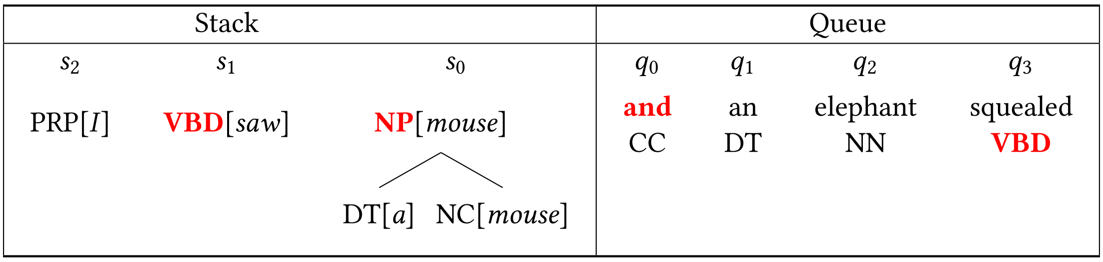


Figure 16: The configuration just before the shift-reduce conflict, with key elements highlighted.

possible here. Finally, the tag of $q_3$ is a key indicator because the presence of a verb in this case suggests that nominal coordination is again not possible. However the verb is not necessarily situated at $q_3$. The presence of a verb later on in the sentence is what is important, but the position of this verb is unbounded to the right, and we cannot identify one particular element of the queue at which the verb should be situated.

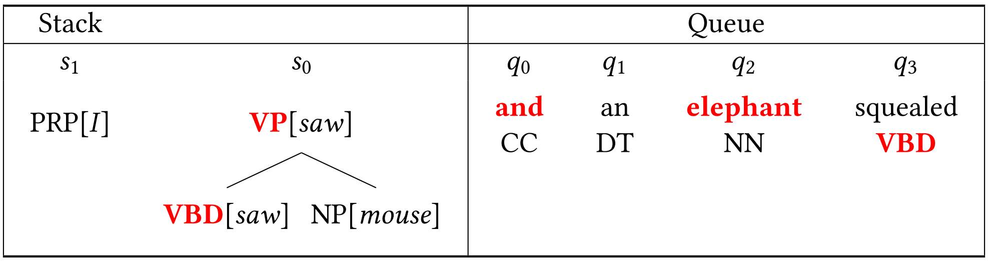


Figure 17: The configuration just before the shift-reduce conflict, with key elements highlighted.

If we were to choose to reduce the two topmost elements, we are again faced with a shift-reduce conflict, of whether to shift the conjunction (to create verbal coordination) or to reduce the topmost stack elements in view to creating sentential coordination. This configuration is shown in Figure 17. Again it may be possible to suggest key trigger elements, for example the verb phrase at $s_0$, the noun at $q_2$, the token “and” at $q_0$ and the verb at $q_3$. However the situation is far from clear. It is difficult to predict the position of the elements and the problem is often unbounded and too complex to model with position numbers, as must be done here.

**The problem.** Especially in the case of coordination, which is not a very intuitive linguistic phenomenon to capture in terms of features, the task of manually selecting configuration elements and values is an incredibly difficult task. It is sometimes possible, as with PP-attachment to identify certain key elements, but the interaction of these elements with other key elements for other linguistic phenomena makes the overall task very difficult and time-consuming. It also necessitates having an in-depth knowledge of the specific annotation guide used for the data used, as well as adapting the knowledge to specific languages.

With so much complexity in the templates, it is often difficult to predict which templates will be useful. Linguistic intuitions can even sometimes be misleading; a template may be statistically pertinent even though linguistically it is not clear why, and on the other hand a template that might appear linguistically interesting may prove not to be useful. This could also be linked in certain cases to correlation between templates. If two templates provide the same information, then it is not necessary to include them both in the model; it goes against the principle of finding the most compact model. It is also important to take into account the fact that statistical models often use features containing several conditions. Although it is possible to have good linguistic intuitions about features relative to the most local elements and to which pieces of morphological information are most useful, the interaction of several conditions means that choosing the most pertinent templates often goes beyond linguistic intuitions.

### 2.2.2 Automatic selection methods

There has therefore been a very strong interest in developing automatic methods of feature selection, using heuristic methods to approximate the optimisation of the model. Using heuristics means that there is still no guarantee of finding an optimal solution, but automatic methods provide a greater feasibility of nearing it than manual methods, because of their power of calculation and their ability to capture statistical generalisations that go beyond linguistic intuitions.

There are two main families of feature selection method: filter and wrapper methods, which differ in their specificity to a particular learning algorithm. Filter methods use heuristics related to the intrinsic characteristics of the data, whereas wrapper methods integrate the learning algorithm to guide the search of the feature set.

#### 2.2.2.1 Filter methods

Filter methods are a popular method for feature selection in classification tasks due to their efficiency and simplicity. They select features based on general characteristics of the data and follow the principle that such decisions can be made without having to study the specificity of features to a particular task. They are therefore generally much faster than methods that require integrating the learning algorithm (see Section 2.2.2.2 for wrapper methods). Selection is generally made up of two steps: the first a ranking of features according to a set of criteria based on the general properties of the training data, and the second a filtering of the features according to their rank. The resulting subset should show a high correlation between the predictor variables and the labels, but a low correlation between the predictor variables themselves to avoid redundancy. Methods can be univariate, which means that each type of variable is studied independently of the others, or multivariate, which also takes into account their interdependency. Performing a multivariate filtering can be important given that individually good features may not produce the best combination of features when brought together in a model, and there may also be a high amount of redundancy. However taking into account interdependence can be computationally more expensive, which explains why univariate methods are still a popular choice.

We cite three of the most commonly used filter criteria for ranking variables: the Fisher score, mutual information gain and Relief algorithms.

The **Fisher score** is a univariate method used to evaluate the degree to which a variable's values are similar for instances of the same class and different for instances of different classes, the idea being to assign a higher score to variables that maximise the distance between class labels. The score for a variable $v_i$ is defined as follows:

$$S(v_i) = \frac{\sum_{k=1}^{K} N_k \left(\mu_{ik} - \mu_i\right)^2}{\sum_{k=1}^{K} N_k \rho_{ik}^2}, \tag{2.1}$$

where there are $K$ class labels, $N$ indicates the number of instances of class $k$, $\mu_{ik}$ and $\rho_{ik}$ are the mean and variance of the $i$-th variable for a specific class $k$ and $\mu_i$ is the mean of all values of the $i$-th variable. This method assigns scores to variables individually, whereby they can then be ranked and filtered. But the calculation of informativeness is done independently for each variable and therefore the method cannot take into account the possibility that there be redundant variables amongst the highest scoring, nor the fact that there may be low scoring variables that combined are very informative co-predictors. A solution to this problem is provided by Gu *et al.* (2012), who perform a joint selection of variables to maximise a Generalised Fisher Score, capable of reducing redundancy in selected variables.

Another method of assessing informativeness of variables is to calculate the **mutual information gain** between variables and the class labels. The method is very popular due to its simplicity and efficiency, but again is a univariate method, used to calculate the informativeness of individual variables to the class labels. Information gain between the $i$-th variable and class $C_i$ is calculated as follows:

$$\text{Info\_gain}(v_i) = H(v_i) - H(v_i|C_j), \tag{2.2}$$

where $H(v_i)$ is the entropy of the $i$-th variable and $H(v_i|C_j)$ the entropy of the $i$-th variable given the $j$-th class. Extensions of this method have been suggested, such as the predominant correlation method by Lei Yu (2003), which ranks variables according to their individual informativeness for predicting class labels and then filters the results where correlation between two variables is greater than that of one of the variables to the class label, thus eliminating redundancy.

The third method that will be mentioned here is **Relief** (Kira and Rendell, 1992), which is an instance-based selection method designed to rank variables according to their ability to separate closely positioned instances of different classes. They involve iteratively changing the quality of variables depending on their ability to separate close instances. Relief algorithms are useful for large datasets with many dimensions but again cannot reduce redundancy between highly informative variables.

The advantage of filter methods, namely their efficiency, is linked to the fact that they do not require integrating a particular learning algorithm and can simply base the selection on general data properties. This advantage can also be a disadvantage in that their success heavily depends on the specific dataset and the task they are used for. Different feature selection methods produce the best results on different datasets and there does not exist a one-fits-all feature selection method adapted to produce optimal results on all tasks.

#### 2.2.2.2 Wrapper methods

Wrapper methods for feature selection overcome the limitations of filter methods by integrating the learning algorithm into the selection process to guide the search of the feature space and to terminate the search. For large feature spaces, wrapper methods can be computationally heavy, given that the number of possible models is $2^m - 1$ where m is the number of features, and optimising this problem is NP-hard. Therefore, greedy strategies are often preferred for high-dimensional feature spaces, because of their ability to avoid computing all possible subsets. The principle of greedy strategies is to continually modify the ensemble of features that make up the model until the accuracy of the model no longer increases, or at least does not decrease. The final model should contain a subset of the initial set of possible features and should have an accuracy that is better or equal to all intermediate models constructed during the search.

There are two main strategies that determine the direction of search: forward and backward search, both shown in the generic algorithm in Algorithm 2. In forward search, the initial model contains no features at all, and at each iteration a feature is greedily added until there is no increase in the accuracy of the model. The selected feature is the one amongst the remaining unselected features that, if added, would result in the highest accuracy gain over the current model. In a backward search, the initial model contains all possible features and at each iteration the feature whose removal would result in the lowest accuracy loss over the current model is removed. This is repeated until there is no more accuracy gain. In both cases, the final model is the model produced at the last iteration. The advantage of performing a forward search is that, since a final model often contains a very small subset of possible features, it is often much faster to add features than to remove them from a complete set of features, and in general backward searches require training much larger models. However they are less capable than backward methods of taking into account the interdependence of features. If two features do not individually result in an accuracy gain to the model, they will not be added, even if the combination of the two would result in a high accuracy gain, which is known as co-prediction. The backward model can taken into account these dependencies because in the case of the same two features, removing one would result in a high decrease in accuracy and therefore the feature would remain in the model. We will see in Section 2.3 how variants, including forward-backward methods also exist.

**Algorithm 2** A generic framework for either a forward or backward wrapper method

```
Data = $\{(x_1, y_1), \ldots, (x_D, y_D)\}$

SearchSpace$^{(0)}$ ← INITIALISE_SEARCH_SPACE()
FeaturesInModel$^{(0)}$ ← INITIALISE_MODEL_FEATURES()
Model$^{(0)}$ ← INITIALISE_MODEL()                ▷ A model is a set of $t$ feature-weight pairs

for t from 1 to T do
    Feature$^{(t)}$ ← SELECT_FEATURE(SearchSpace$^{t-1}$, Data, FeaturesInModel$^{(t-1)}$)
    FeaturesInModel$^{(t)}$ ← UPDATE_MODEL_FEATURES(FeaturesInModel$^{(t-1)}$, Feature$^{(t)}$)
    Model$^{(t)}$, acc$^{(t)}$ ← FIT_MODEL(Data, FeaturesInModel$^{(t)}$)
    SearchSpace.remove(Feature$^{(t)}$)

Return Model$^{(T)}$
```

In both methods the stopping criterion can either be a fixed limit to the number of rounds $T$ or a more adapted criterion based on a continued increase in generalisation accuracy for the forward wrapper and a decrease in generalisation accuracy below a certain threshold for the backward wrapper.

The downside of the method is its inefficiency, especially given a large dataset and high dimensionality. This is linked to the fact that in the standard algorithm, to select new features, all remaining features must be fit and compared. If the integrated learning algorithm requires training, the selection procedure can be very slow. Taking the forward wrapper, for $T$ features added to the model out of $M$ possible features, the standard algorithm has to train $N + (N-1) + \ldots + (N-T) = TN - \frac{N(N-1)}{2}$ models. Furthermore, we have seen in Section 1.4.3.2 how training time can be highly dependent on the size of the model. This method requires retraining the entire model at each round, and as a new feature is addeed to the model each time, training time increases at each round.

Hybrid filter-wrapper methods exist, which aim to combine the merits of both methods; the efficiency of filter methods and the specificity of wrapper methods to the task at hand. For example, Zhu *et al.* (2007) combine a univariate filter method with a wrapper method for a genetic algorithm to appropriately add and delete features to and from the search space, based on the ranking provided by a filter method. Das (2001) proposes a filter-based method, using boosted decision stumps to iteratively select features, with training accuracy used as a stopping criterion. However, although these methods do integrate training accuracy, they do so just to determine when to terminate the search, and therefore the search itself is not guided by the learning algorithm and, for more complex algorithms such as those used in parsing, are likely to produce less good results.

## 2.3 Feature selection methods for parsing

As in many domains where the number of potential features can be very large, the parsing community has started to become interested in methods of selecting an optimal subset of features. In many cases (Zhang and Nivre, 2011; Hall *et al.*, 2007), this is done manually by trial-and-error and the use of linguistic intuitions. However, this can be a tedious task and due to the specificity of data-driven parsing models to the domain of the data on which they were trained, it is often necessary to re-perform this selection procedure each time a new language, style or domain is to be used for training. Furthermore, the optimal feature subset is often very specific to a particular parsing algorithm or to the parameters of the algorithm, meaning that the optimal features are not necessarily transferable from one system to another.

We have seen in Section 1.2 how the growing interest in languages other than English, and in particular morphological rich languages has led to the rise in popularity of discriminative models, which can more easily integrate a large number of different lexical features. In search of expressivity, the authors of these models now have at their disposition a very large number of potential features and enumerating all possible models has become unfeasible. There has therefore been a certain amount of interest in feature selection specific to parsing. Due to the fact that parsing is not a simple classification task but structured prediction, simple filter methods are poorly adapted for selecting an optimal subset and so methods rely rather on the wrapper approach, integrating the parser's performance and training into the feature selection method. However training a parser can be particularly time-consuming, especially when large amounts of high-dimensional data are used, and methods therefore need to be orientated towards efficiency. They incorporate heuristics to limit the time taken for feature selection either by adapting the search method or by reducing the search space. Most automatic methods for parsing are therefore greedy methods, performing variants of backward and forward selection.

Whilst the majority of the methods concentrate on a forward wrapper method, due to the fact that training very high-dimensional models (a necessity for backward selection) is very time inefficient, some backward methods do exist, provided that the total number of templates is reasonably small to start with. Attardi *et al.* (2007) perform feature selection on their shift-reduce dependency parser by starting out with a model with 43 templates and performing backward selection from this initial subset to remove one template. The advantage of this approach is that they need not perform multiple rounds of the feature selector, dramatically reducing the time necessary for selection and do succeed in producing different models for ten different languages, improving the labelled attachment score (LAS) up to 4% for some languages in comparison to the scores obtained on the default set. However this method relies on the fact that they use a small initial subset and perform very little feature selection within it.

This can be compared to the entirely forward method used by Nilsson and Nugues (2010), also for shift-reduce dependency parsing, in which the initial model contains only the first word of the first queue element and forward selection is performed for all other possible features. The attributes taken into account are the wordform, the part of speech and the dependency label. To reduce the size of the search space, the search is guided by the order of stack and queue elements, under the informed assumption that elements more local to the focus tokens result in a greater increase in parsing performance than elements further away. They achieve state-of-the-art performances for English and Swedish with smaller final models than most state-of-the-art models.

Other methods use a mixture of forward and backward methods to combine the advantages of the two approaches. As in Attardi *et al.* (2007), Ballesteros and Nivre (2014) start with an initial default set of features and perform backward and forward selection of features as the feature selection step in MaltOptimizer, an optimiser for the dependency parser MaltParser. Given that the feature set is large and training is time-consuming, they fix heavy constraints and heuristics on the order in which features are processed, and on how backward and forward selection is performed. The process is formed of six steps:

1. Backward selection of POS features (starting from elements furthest away) and then forward selection if none were eliminated
2. Backward selection of wordform features and then forward selection if none were eliminated
3. Backward selection of dependency link features and then forward selection of POS features for the dependency tree nodes
4. Forward selection of contextual features using POS and wordform features
5. Forward selection of coarse POS tags, morphosyntactic features and lemma features starrting with the nearest elements to the focus tokens and working outwards.

These heuristics have been carefully designed to intelligently guide the search, which would otherwise be too computationally expensive. Although the use of a heuristic search means that there are no optimisation guarantees, they found that in the majority of cases for a variety of languages, the optimised parser produced slightly higher LAS scores than carefully selected manual feature models, with an average of 0.58% increase in LAS over manual models for ten different languages, proving that automatically selected models can be high performing. They notice a big difference in LAS scores between the initial default model and the final resulting model for a number of languages, including Basque, Hungarian, Czech and Turkish, whilst the improvements for Catalan, Japanese, Chinese and Swedish were more modest. The authors explain this variation in terms of availability of training data (for some languages more morphological clues are available) and also on the choice of a non-projective or projective dependency algorithm.

Ballesteros and Bohnet (2014) extend the ideas used in MaltOptimizer to deal with a larger number of feature templates. They also perform feature selection on the Mate parser, and test two models, one purely transition-based and the other transition-based with a graph-based completion model. For both models their search space instantiates values from stack and queue elements as described in Section 1.4. As in MaltOptimizer, they start from a default set of features, attempt to perform backward selection on each of the features in this initial set, and then perform standard forward selection for the remaining features, adding each feature if the gain is above a certain threshold. Again, they rely on heuristics to avoid testing feature templates pertaining to elements and values further away from focus tokens if features involving more local variants were excluded. They achieve state-of-the-art performances for English, Chinese and Russian and results that are comparable with Bohnet *et al.*'s (2013) results for Hungarian, but using fewer features.

He *et al.* (2013) also implement a feature selection strategy for discriminative parsing, but for a graph-based (MST) model. Their strategy to increase the efficiency of the selection method is to apply feature selection dynamically, depending on the necessity to add more templates. Their search space includes 268 first-order feature templates and 112 second-order templates. They start by ranking all features in a forward wrapper-style way and then grouping them into a certain number of groups, such that each group results in a similar accuracy gain. The parser is run a certain number of times and at each round, a parse tree is produced. For each link in the parse tree, a binary classifier decides whether the link is reliable enough to be locked for the final model or whether another template group must be added to decide on the link. In the next round, locked links remain and are treated as constraints for the other decisions made in the tree, and the new links produced are again classified as lock or add. After a certain number of rounds, a locked tree is produced, without having had to use all templates for all links. This approach has the advantage of only using templates when they are needed, thus reducing the calculation times for parsing, but is rather specific to graph-based models.

CHAPTER 3

# OUR APPROACH: MODEL SELECTION VIA BOOSTING

In the case of the constituency-based shift-reduce parser throughout this thesis, the majority of problems faced by the feature selection methods for dependency parsing (mentioned in the previous section) are the same, due to the fact that they use very similar algorithms. However we are potentially faced with a dimensionality problem that surpasses all of the previously mentioned feature selection methods for parsing, due to the fact that the ultimate aim is to test the hypothesis that supplementary morphological information can be useful for parsing morphologically rich languages. We are therefore likely, at least at a later date, to include more morphological information than most of the methods described previously. We face a search space in the order of several thousand, or up to tens of thousands for data for up to six different types of lexical attribute (see Section 2.1.2.1 for an explanation of this quantification).

The methods previously mentioned use a variety of heuristics to guide the feature search, including starting from a default set of feature templates, regrouping templates into groups or applying more vigorous constraints on the order and direction of feature selection. However they do not tackle one of the major reasons for the extremely long duration of most feature selections in parsing, which is the time needed to train models. The standard forward wrapper, in which the template that would contribute the highest gain in accuracy is added to the model at each iteration, requires retraining an ever-growing model; the greater the size of the model, the longer training times will be. We propose a solution to this problem by suggesting a method in which feature selection is constant at each round. This can be achieved by performing a stepwise additive fit, which means that the model can be constructed template-by-template without ever changing the coefficients of the templates already added. Therefore, whilst the same number of models need to be trained at each step of the algorithm, the model trained contains only the template that would be added, which drastically reduces the time needed for training.

An algorithm that has these desired properties is the boosting algorithm AdaBoost. Boosting algorithms are ensemble methods, originally designed for predictive modelling, used to combine a set of weak learners[1] to produce a strong one, by additively and iteratively choosing the best weak learner and assigning a coefficient (the classifier's importance in the final model) until no more weak classifiers are available. The weighted combination of the classifiers should produce a superior classifier to each individual one. Boosting can be seen as a feature selection method in that it iteratively selects the best weak classifiers until there are no more to add or until a certain number have been added, with the result that the final subset produces a strong model, as in a forward wrapper feature selection method. It has previously been used for feature selection for automatic face-detection in the domain of imagery (Viola and Jones, 2004), for which the very large number of possible features necessitates a very efficient feature selection method, and for a variety of classification tasks by Das (2001) using boosted decision stumps. However to our knowledge, boosting methods have not yet been used in the context of feature selection for parsing. In our case, the weak learners would be feature templates or groups of templates, combined to form an ensemble of templates that can be used to retrain a strong parser.

## 3.1 The AdaBoost algorithm

Figure 3 shows the algorithm for the standard two-class AdaBoost. A weight distribution over the training examples, initially an even distribution, is updated at each iteration. This is a way of encoding the dependency of the choice of the next weak classifier on those already in the model. We will see in the paragraph 'Minimising an exponential loss function' how these weight updates are actually a simple way of encoding the exponential loss function in the data in order to minimise this loss function.

At each round $t$, each of the possible weak classifiers is fit to the weighted data and the weak classifier with the smallest weighted error is selected. A coefficient $\alpha$ for the weak classifier, representative of its weight in the final fit, is calculated as half the log odds of the weighted error. This coefficient $\alpha$ is then used to update the weights of each example, therefore giving more weight to misclassified examples. This has the effect of selecting classifiers that concentrate on the "hard" examples, i.e. those that were misclassified by the weak classifiers already added.

The final boosted prediction is a weighted prediction of the weak classifiers added to the fit, each weighted by its coefficient $\alpha$.

[1] A weak learner (also known as a weak classifier or weak hypothesis) is a classifier that performs slightly better than random classification. In the two-class case, it is a classifier with an accuracy of just above 50%. An example of a weak classifier is a decision stump (a one-level decision tree)

**Algorithm 3** Standard two-class AdaBoost (Freund and Schapire, 1999)

```
                                              ▷ Data = $\{(x_1, y_1), \ldots, (x_D, y_D)\}$
Initialisation of data weights
        $w_i^{(1)} = \frac{1}{D}$, i = 1, 2, ..., D
for t=1 to T do
    (i)   Fit classifier $g^{(t)}$(x) to data using weights $w_i^{(t)}$
    (ii)  Calculate the weighted error
        $err^{(t)} = \frac{\sum_{i=1}^{D} w_i^{(t)} \mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right)}{\sum_{i=1}^{D} w_i^{(t)}}$
    (iii) Calculate $\alpha^{(t)}$
        $\alpha^{(t)} = \frac{1}{2} \log\left(\frac{1 - err^{(t)}}{err^{(t)}}\right)$
    (iv)  Update data weights using $\alpha^{(t)}$
        $w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(-\alpha^{(t)} y_i\, g^{(t)}(x_i)\right)$, i = 1, 2, ..., D
          And normalise weights such that $\sum_{i=1}^{D} w_i^{(t+1)} = 1$
end for
Prediction is the sign of a weighted prediction of models $g^{(t)}$, t = 1, 2, ..., T
        $f$(x) = sign$\left(\sum_{t=1}^{T} \alpha^{(t)} g^{(t)}(x)\right)$
```

## 3.2 AdaBoost as forward stagewise additive modelling

As demonstrated in Hastie *et al.* (2009), Zhu *et al.* (2006) and Friedman *et al.* (2000), boosting is equivalent to a forward stagewise additive modelling, in which each mini-model is fit successively, optimising an exponential loss function and encoding the performance of models already added to the fit in the reweighting of the examples. Here we present the statistical view of boosting as presented in Hastie *et al.* (2009).

**Why use stagewise additive modelling?** In many optimisation problems, finding the analytical solution can be very computationally expensive, especially when multiple feature functions are combined to produce a single optimal model. A solution to this problem is to approximate the solution to the optimisation problem by proceeding in a forward additive stepwise fashion, i.e. adding the feature functions one by one without having to recalculate the coefficients of the functions already added to the model.

Take a linear regression model $f$(x), made up of a series of T base functions $g^{(t)}(x)$, each weighted by a coefficient $\alpha^{(t)}$:

$$f^{(T)}(x) = \sum_{t=1}^{T} \alpha^{(t)} g^{(t)}(x). \tag{3.1}$$

If the loss function to be minimised is least squares, which for one example is calculated by

$$L\left(f^{(T)}(x_i), y_i\right) = \left(y_i - f^{(T)}(x_i)\right)^2, \tag{3.2}$$

the global loss function to be optimised, with $D$ examples and $T$ base functions in the model is

$$\min_{\{(g^{(t)},\, \alpha^{(t)}),\, 1 \leq t \leq T\}} \sum_{i=1}^{D} L\left(\sum_{t=1}^{T} \alpha^{(t)} g^{(t)}(x_i), y_i\right), \tag{3.3}$$

which amounts to finding the combination of functions and their associated coefficient $\alpha$ that minimise the least squares loss function. For a large number of possible functions, this calculation necessitates complicated optimisation techniques. A solution is to fit each function and its coefficient separately and successively, without modifying the coefficients of those already included in the model, but by memorising the residual losses of the additive model in order to choose the next best base function. This method is known as a stagewise (or stepwise) method, because it progressively approaches the minimum of the loss function as each new base function is added. Therefore at each step, the optimisation only needs to be performed for one base function $g^{(t)}$ and its coefficient $\alpha^{(t)}$.

The demonstration is simple: from Equation 3.1, we can see that a model $f^{(T)}(x)$ can be obtained from an addition of the preceding model $f^{(T-1)}(x)$ (containing basis functions 1 to $T-1$) and the $T$th base function, weighted by its coefficient $\alpha^{(T)}$, as shown in Equation 3.4.

$$f^{(T)}(x) = f^{(T-1)}(x) + \alpha^{(T)} g^{(T)}(x). \tag{3.4}$$

From this, we can extend this principle to the calculation of the loss function, by plugging Equation 3.4 into the loss function in Equation 3.2 as follows:

$$\begin{aligned} L\left(f^{(T)}(x_i), y_i\right) &= \left(y_i - \left(f^{(T-1)}(x_i) + \alpha^{(T)} g^{(T)}(x_i)\right)\right)^2 \\ &= \left(y_i - f^{(T-1)}(x_i) - \alpha^{(T)} g^{(T)}(x_i)\right)^2, \end{aligned} \tag{3.5}$$

in which the optimisation uses the residual loss $y_i - f^{(T-1)}(x_i)$ (i.e. the loss obtained for the example using all features already added to the model) to encode the dependency on already added features. By keeping track of the residual loss of the base functions already added to the fit, the next function to be added can therefore be more easily computed in order to approximate the solution to our original optimisation problem in Equation 3.3.

**Minimising an exponential loss function.** The loss function in the standard binary AdaBoost is not least squares but the exponential loss function shown in Equation 3.6.

$$L\left(f(x_i), y_i\right) = \exp(-y_i\, f(x_i)) \tag{3.6}$$

There are two main advantages of using an exponential loss function. The first is that it is easy to derive (see the derivation in Appendix A) and makes it possible to represent the minimising of loss as a minimisation of the loss of a weighted sample, which results in the simple AdaBoost algorithm based on weighted updates, as shown in Algorithm 3. This also means that more weight is given to misclassified examples, which has the effect of finding classifiers that concentrate on "hard" examples. The second advantage is the observation that minimising the exponential loss is equivalent to estimating a logistic regression model in a stagewise fashion, as demonstrated by Friedman *et al.* (2000)'s statistical explanation, which means that it is equivalent to a Bayes classifier, minimising misclassification error.

AdaBoost is a forward additive model, and as with the minimisation of the least squares function (discussed in the paragraph 'Why use stagewise additive modelling?'), the exponential loss based on the training data can be minimised incrementally by gradually improving on residual losses. The same additive property as in Equation 3.5 can therefore be seen for exponential loss as shown in Equation 3.7.

$$L\left(f^{(T)}(x_i), y_i\right) = \exp\left(-y_i\left(f^{(T-1)}(x_i) + \alpha^{(T)} g^{(T)}(x_i)\right)\right) \tag{3.7}$$

At step $T$, we are therefore looking for the base function and its coefficient that minimise the loss, which is based on the residual loss of the model produced at step $T-1$ plus that of the base function, weighted by its coefficient, as shown in Equation 3.8.

$$\begin{aligned}\left(\alpha^{(T)}, g^{(T)}\right) &= \underset{(\alpha,g)}{\operatorname{argmin}} \sum_{i=1}^{D} \exp\left(-y_i\left(f^{(T-1)}(x_i) + \alpha\, g(x_i)\right)\right)\\ &= \underset{(\alpha,g)}{\operatorname{argmin}} \sum_{i=1}^{D} \mathbf{exp}\left(\mathbf{-y_i\, f^{(T-1)}(x_i)}\right) \exp\left(-y_i\, \alpha\, g(x_i)\right)\\ &= \underset{(\alpha,g)}{\operatorname{argmin}} \sum_{i=1}^{D} \mathbf{w_i^{(T)}} \exp\left(-y_i\, \alpha\, g(x_i)\right)\end{aligned} \tag{3.8}$$

The residual loss, shown in red, can be seen as the weight of each example $w_i$, since it is independent of both $g(x)$ and $\alpha$. The weights are updated at each iteration $t$ to encode the fact that the residual losses of an additive model are taken into account when choosing the next best feature.

As long as the value of $\alpha$ is positive, we can easily show that

$$g^{(T)} = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{D} w_i^{(T)}\ \mathbb{I}\left(y_i \neq g(x_i)\right), \tag{3.9}$$

which means that the next best $g$ is the one that minimises the weighted error rate on the training data.

The function can be derived to calculate the coefficient of $\alpha$ that minimises the function, the solution being the half the log odds of the error, as seen in Equation 3.10.

$$\alpha = \frac{1}{2}\log\frac{1-err}{err} \tag{3.10}$$

The proof behind these two observations is described more fully in Appendix A

## 3.3 Multi-class AdaBoost

Freund and Schapire (1999) propose a simple extension (AdaBoost-M1) of their original algorithm to the multi-class case by performing a one-vs.-all binary classification on each of the classes. The obvious advantage of this method is that the binary AdaBoost can be used with very little modification. However this has the limitation of constraining weak learners to having an error rate < 0.5, which for a multi-class case is more challenging than in the binary case, especially when there a large number of classes.

Here we will study the use of the multi-class variant SAMME (Stagewise Additive Modelling using a Multi-class Exponential loss function), as described in Zhu *et al.* (2006). This variant has the advantage, as do several others, of reducing to the standard binary AdaBoost if there are only two classes. What is more, weak learners only need to be better than random guessing at $\frac{1}{K}$ where $K$ is the number of classes, rather than the stricter $\frac{1}{2}$ necessary for Adaboost-M1.

**Algorithm 4** Multi-class (SAMME) AdaBoost (Zhu *et al.*, 2006)

```
                                                        ▷ Data = $\{(x_1, y_1), ..., (x_D, y_D)\}$
Initialisation of data weights
        $w_i^{(1)} = \frac{1}{D}$, i = 1, 2, ..., D
**for** t=1 to T **do**
    (i)  Fit classifier $g^{(t)}$(x) to data using weights $w_i^{(t)}$
    (ii)  Calculate the weighted error
        $err^{(t)} = \frac{\sum_{i=1}^{D} w_i^{(t)} \mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right)}{\sum_{i=1}^{D} w_i^{-t)}}$
    (iii)  Calculate $\alpha^{(t)}$
        $\alpha^{(t)} = \log\left(\frac{1 - err^{(t)}}{err^{(t)}}\right) + \log(K-1)$
    (iv)  Update data weights using $\alpha^{(t)}$
        $w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(\alpha^{(t)} \cdot \mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right)\right)$, i = 1, 2, ..., D
        And normalise weights such that $\sum_{i=1}^{D} w_i = 1$
**end for**
Prediction is the sign of a weighted prediction of models $g^{(t)}$, t = 1, 2, ..., T
        $f$(x) = argmax$_k$ $\sum_{t=1}^{T} \alpha^{(t)} \mathbb{I}\left(g^{(t)}(x) = k\right)$
```

The algorithm is shown in Algorithm 4. SAMME is in fact very similar to the two-class AdaBoost (shown in Algorithm 3). A distribution over training examples is kept at each round, encoding the exponential loss of previously added weak classifiers. Apart from the prediction, which is now an argmax over possible classes, the element that changes is the calculation of the coefficient $\alpha$ and the update of weights. The calculation for $\alpha$ for the multi-class version is now the log odds of the error plus an additional term $\log(K-1)$ rather than being half the log odds as for the two-class version. As explained in Zhu *et al.* (2006), the term $\log(K-1)$ has theoretical justifications. In their explanation, they start by presenting a notational variant of the two-class AdaBoost, which resembles the multi-class version and is therefore used as its basis. The first step is identifying the notational equivalence between $-y_i\, g^{(t)}(x_i)$ and $2\mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right) - 1$. We can therefore change the weight update in the two-class case to

$$(3.11)\quad w_i^{(t+1)} = w_i^{(t)} \cdot \exp\left(\alpha^{(t)} \cdot \mathbb{I}\left(y_i \neq g^{(t)}(x_i)\right)\right), \text{i} = 1, 2, ..., \text{D},$$

which, being half the update of the original version, requires doubling the coefficient $\alpha$, making the calculation of the coefficient now the log odds of the error, rather than half the log odds, as follows:

$$(3.12)\quad \alpha^{(t)} = \log\left(\frac{1 - err^{(t)}}{err^{(t)}}\right)$$

This notation has the merit of abstracting away from the two-class case and the use of {+1, -1}. Now the only difference between the two-class case and SAMME is the term $\log(K-1)$ added to the log odds of the error. The reason behind this addition is that now the weak classifier need only have above random classification rate ($\frac{1}{K}$), rather than $\frac{1}{2}$ as in Freund and Schapire's original extension to the multi-class case. We can see that in the case of a random classifier (one with error rate as $1 - \frac{1}{K}$), the coefficient $\alpha$ is equal to zero. This also has the advantage of reducing to the two-class algorithm in the case where $K = 2$.

## 3.4 Adapting boosting to feature selection for parsing

Ensemble methods such as boosting can be a popular choice for classification tasks because of their ability to combine weak classifiers to produce a stronger one, but, as evoked by Cortes *et al.* (2014), these traditional methods pose problems for structured prediction, such as in parsing. Due to the fact that in structured prediction, predicted outputs are sequences of classes and not individual classes, weak classifiers are of course evaluated on their ability to predict the sequence of classes. This means that regardless of the fact that a weak learner may be a local expert and have very high performances at predicting a certain element of the sequence, if globally it is not a high performer, it will always lose out to a more globally successful one. Models that are on average high-performing will therefore always triumph over those models that could have provided expert local advice. Cortes *et al.* (2014) try to solve this problem by using a DAG to represent the individual substructures that make up the sequence. They keep a weight distribution over paths, enabling the problem to be reformulated as finding the best path expert in the DAG. Figure 18 shows a representation of this DAG for a structured example containing $L$ substructures and $p$ weak hypotheses. Choosing the best path expert means that it is possible to combine the different predictions

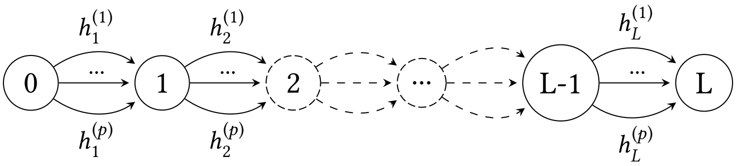


Figure 18: DAG structure for structured prediction (Cortes *et al.*, 2014)

of each substructure rather than taking a single path for one given weak hypothesis.

However, the effective use of ensemble methods for structured prediction is known to be particularly problematic, notably in terms of calculating optimisation guarantees. In the case of parsing, several problems arise if we wanted to implement a boosting algorithm for global optimisation. The first is knowing how many different classes to consider, as this number is necessary to calculate the coefficient $\alpha$ and to ensure that classifiers added to the final fit indeed have better than random accuracy. In theory, since classes are sequences of actions, there can be an infinite number of classes because sentences are not limited in length. However this is very impractical and problematic for the algorithm, notably because there would be no lower limit for the performance of a weak learner. A simple solution could be to fix the number of classes simply as the number of different sequences in the training data. In the worst case scenario, the number of classes would be equal to the number of sentences, and therefore a weak classifier would be one that is able to correctly predict at fewest two sequences of actions. This leads to the second problem, which is that it may be difficult in some cases to find a sufficient number of weak classifiers, especially if each weak classifier is a very small one, containing only one or two features.

**Local feature selection and global refitting.** For these reasons, we choose to use boosting to select features based on a simple local optimisation, for which examples are not sequences, but simple pairs $(x_i, y_i)$ where $x_i$ is a configuration and $y_i$ is an action. A similar approach by Wang *et al.* (2007) shows surprisingly good results on English and Chinese from boosting locally optimised weak parsers (locally trained), but for which the error and the coefficient are calculated according to the structured output error. We therefore choose to perform boosting using a local optimisation with the unique aim of performing feature selection. We will then use the resulting selected templates to retrain a global model on the same data to ensure that the model is a good fit for structured prediction. This method, though an approximation of global optimisation, has the advantage that each weak classifier can be trained much faster due to the fact that fewer training rounds are necessary for perceptron convergence.

# CHAPTER 4 EXPERIMENTS

## 4.1 Methods

To evaluate our approach to feature selection by boosting, we shall test and compare the method to a naive forward wrapper algorithm, concentrating on both efficiency of the feature selection and the generalisation performance of the resulting model. Both methods require the use of additional heuristics to reduce the size of the feature space, and so we shall test two different heuristics.

The first method, our baseline in terms of running time, which we will call **method A1**, is the standard forward wrapper method (see Section 2.2.2.2), using a heuristic by regrouping templates into back-off style template groups. We expect this method to be high performing but very inefficient. It therefore serves as a baseline on which to improve efficiency and a target to reach in terms of its generalisation performance. We compare this method to two methods by boosting: **method B1** uses the same template grouping heuristic as method A1, and **method B2** uses an additive search space heuristic. As an approximation of the optimisation of the standard forward wrapper, we are aiming for generalisation performances as high as the method A1, but in a much faster running time. The methods are summarised in Table 3. Both heuristics will be described in further detail below.

| | Heuristic | |
|---|---|---|
| Method | Template grouping | Additive search space |
| Standard Wrapper | A1 | - |
| Boosting | B1 | B2 |

Table 3: The three methods tested

Note that method A1 was not tested with the heuristic of an additive search space. The additive search space uses weak classifiers with only a single template and the standard wrapper method relies on a global optimisation, which produces particularly poor and sometimes incoherent results when trained on such a weak model. This same effect is not seen however with the local optimisation of the boosting method.

## 4.2 Data

All methods will be tested using data from the French Treebank (Abeillé *et al.*, 2003), a corpus of journalistic texts from the French newspaper *Le Monde*. We use three versions of the corpus, which we refer to as FTB$\alpha$, FTB$\alpha \leq 20$ and FTB$\beta$.

**FTB$\alpha$ and FTB$\alpha \leq 20$.** The two FTB$\alpha$ sets are derived from the version of the FTB produced for the 2013 SPMRL shared task (Seddah *et al.*, 2013). The data, used to train the models presented in Crabbé (2014), contains little morphological information, only the wordform, the POS-tag and an extra value ‘sword’, short for ‘smoothed word’, in which the wordform is replaced by a symbol $UNK$ for wordforms with fewer than 2 occurrences in the corpus. An example of the format can be found in Figure 19. The difference between the FTB$\alpha$ and FTB$\alpha \leq 20$ is that FTB$\alpha$ contains all sentences provided for the shared task, whilst FTB$\alpha \leq 20$ is only a subset, containing sentences of 20 tokens or fewer.

```
                     word            tag             sword
<ROOT−head>
    <SENT−head>
        <VN−head>
                     Est             V−head          Est
                     −ce             CLS             −ce
        </VN>
                     déjà            ADV             déjà
        <NP>
                     le              DET             le
                     cas             NC−head         cas
        </NP>
        <PP>
                     chez            P−head          chez
            <NP>
                     les             DET             les
                <NC+−head>
                     marins          NC−head         $UNK$
                     −               PONCT           −
                     pêcheurs        NC              pêcheurs
                </NC+>
            </NP>
        </PP>
                     ?          PONCT      ?
    </SENT>
</ROOT>
```

Figure 19: An example of the FTB$\alpha$ and FTB$\alpha \leq 20$ datasets in the native parser format

**FTB$\beta$.** FTB$\beta$ contains more morphological features than FTB$\alpha$ and FTB$\alpha \leq 20$ and will allow us to test the hypothesis that extra morphological information can be useful in parsing French. The data is derived from the version of the FTB produced for the 2014 SPMRL shared task Seddah *et al.* (2014) and uses the morphological CRF tagger Marmot (Müller *et al.*, 2013) for a better prediction of POS tags. The dataset contains six lexical columns: the wordform, the POS-tag and four extra columns: 'mwe' indicating multi-word tokens and their position in the word, using the BIO format, 'num' indicating grammatical number, 'mood' for grammatical mood and 'gen' indicating grammatical gender. This morphological information was produced automatically using Marmot, trained on FTB data using a 10-fold jackknifing technique (explained in Crabbé (2015)). An example of FTB$\beta$ is given in Figure 20.

```
                     word          tag        mwe     num        mood       gen
<ROOT-head>
    <SENT-head>
        <VN-head>
                     Est           V-head     O       s          ind        Na
                     -ce           CLS        O       s          Na         m
        </VN>
                     déjà          ADV        O       Na         Na         Na
        <NP>
                     le            DET        O       s          Na         m
                     cas           NC-head    O       s          Na         m
        </NP>
        <PP>
                     chez          P-head     O       Na         Na         Na
            <NP>
                     les           DET        O       p          Na         m
                <NC+-head>
                     marins        NC-head    B_NC    p          Na         m
                     -             PONCT      I_NC+   Na         Na         Na
                     pêcheurs      NC         I_NC+   p          Na         m
                </NC+>
            </NP>
        </PP>
                      ?          PONCT        O       Na         Na         Na
    </SENT>
</ROOT>
```

Figure 20: An example of the FTB$\beta$ dataset in the native parser format

Due to the fact that method A1, the baseline in terms of selection time, is particularly inefficient and therefore unfeasible for the complete dataset (both FTB$\alpha$ ad FTB$\beta$), all methods will be trained and compared on FTB$\alpha \leq 20$. Once the model selection is complete, the resulting templates from each method will be used to retrain a global model on the FTB$\alpha$ dataset to test the generalisability of the features to more complete data. In addition to these comparisons, method B2 can also be run on more complete and complex data, since it is expected to be far more efficient than the other two methods. Method B2's performance on the FTB$\alpha \leq 20$ dataset can therefore be compared to its performance on the FTB$\beta$ dataset.

## 4.3 Results

### 4.3.1 Method A1 (standard forward wrapper with template grouping)

**Heuristic: Template grouping.** This principle of regrouping features can be extended to the regrouping of different templates into classes of template. There are different methods by which this could be possible, but the method we use relies on the fact that templates contain a certain number of conditions (up to three) and that therefore one template can therefore be said to subsume another if all conditions in the second are also contained in the first. We refer to these groups as template groups. Each template group initially contains seven templates: three uni-conditional, three bi-conditional and one tri-conditional template. Whilst uni-conditional and bi-conditional templates can belong to several template groups, the tri-conditional template has a unique membership to a template group and can therefore be said to define the group. The aim of this regrouping is to include templates in a back-off style, with templates with similar conditions in a same group. For example, the template 'T[2] = q0.word & q1.word & q2.word' subsumes the following templates:

T[3] = q0.word & q1.word
T[4] = q0.word & q2.word
T[5] = q1.word & q2.word
T[6] = q0.word
T[7] = q1.word
T[8] = q2.word

Templates 2-8 can therefore be regrouped into a single set of templates, to be selected all together, thus reducing the number of objects in the search space to the number of templates with three conditions. As template groups are added to the final fit, the uni- and bi-conditional templates included within it are removed from the remaining template groups so as not to include a template more than once in the final fit.

**Feature selection.** Method A1 was run for six iterations and stopped manually due to time constraints. The progression of the feature selector on the FTB$\alpha \leq 20$ dataset is shown in Table 4.

| Iter | Dev acc | Dev acc gain | Tpls added | Model size | Time (hrs) | Total time (hrs) |
|---|---|---|---|---|---|---|
| 1 | 0.807 | 0.807 | 7 | 7 | 68 | 68 |
| 2 | 0.856 | 0.049 | 7 | 14 | 221 | 289 |
| 3 | 0.864 | 0.008 | 7 | 21 | 277 | 566 |
| 4 | 0.869 | 0.005 | 7 | 28 | 297 | 863 |
| 5 | 0.871 | 0.002 | 6 | 34 | 301 | 1163 |
| 6 | 0.872 | 0.001 | 5 | 39 | 334 | 1497 |

Table 4: Model characteristics from method A1 at each round of the feature selector.

We can see that with each iteration, the accuracy on the development set increases by a diminishing score each time. The heuristic used is the grouping of templates and for the first four iterations the template group added has the maximum template group size of seven templates, the fifth iteration adding six templates and the sixth only five. Since at each iteration the model trained increases in size, the time taken to perform each iteration increases, despite the fact that with each iteration the search space decreases by one template group. This is due to the fact that more calculations are necessary for a larger model and because a larger model takes longer to converge. With this method we approximated a termination of the perceptron training on convergence by stopping training when the loss change was below a certain value for five consecutive rounds, and in all cases, the maximum number of perceptron iterations was limited to 35.

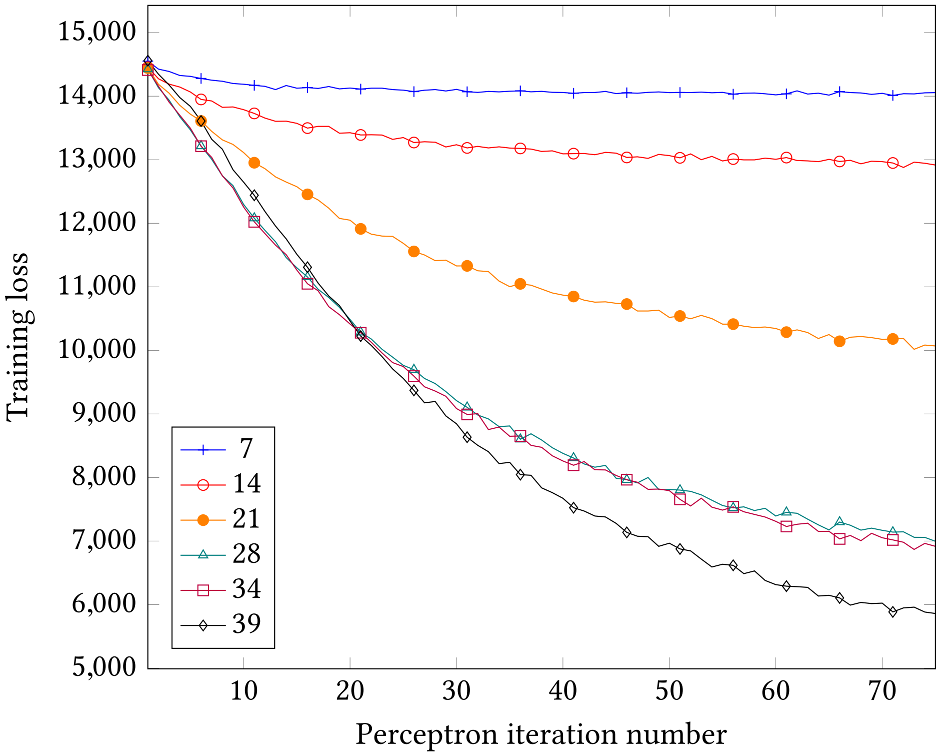


Figure 21: Convergence of loss during perceptron training for method A1 on FTB$\alpha$ data, shown at each iteration of the feature selector: with 7, 14, 21, 28, 34 and 39 templates.

To test the fact that the larger models take longer to converge, we plot the training logs of the intermediate and final models in Figure 21. We observe that in all cases, perceptron training drives down the training loss, and more so for larger models. It is interesting to note that there appears to be little difference between the 28-template model and the 34-template model, perhaps implying that the adding of this final template group changes little in terms of performance. The next round, with 39 templates shows a greater decrease in the loss, suggesting that the interactions between these templates added and the ones in the previous round add a greater expressivity to the model. We can also see that smaller models converge must faster; the smallest model appears to reach convergence after approximately 10 iterations, whereas the three largest models do not appear to have converged even after 75 iterations.

**Accuracy on the development set.** We retrained the model produced after each iteration of the feature selection on the FTB$\alpha$ training set, using a global optimisation, a beam size of 8 and early updates.

| | | F-score with punctuation | | | F-score without punctuation | | |
|---|---|---|---|---|---|---|---|
| Iter | Size | FTB$\alpha \leq 20$ | FTB≤40 | FTB$\alpha$ | FTB$\alpha \leq 20$ | FTB≤40 | FTB$\alpha$ |
| 1 | 7 | 84.88 | 75.80 | 74.05 | 85.93 | 79.16 | 75.80 |
| 2 | 14 | 87.98 | 80.57 | 78.88 | 88.89 | 84.12 | 80.57 |
| 3 | 21 | 88.98 | 82.31 | 80.41 | 90.00 | 85.09 | 82.31 |
| 4 | 28 | 89.44 | 83.03 | 81.28 | 90.52 | 85.63 | 83.03 |
| 5 | 34 | **89.89** | **83.50** | **81.77** | **91.03** | **85.95** | **83.50** |
| 6 | 39 | 87.73 | 79.72 | 79.72 | 88.91 | 84.80 | 81.57 |

Table 5: Method A1: F-score on the FTB$\alpha$ development set, having retrained the model produced after each round of the feature selector.

Table 5 shows the F-scores after retraining the model on the FTB$\alpha$ data set using the templates produced at each step of the feature selector. We see a general increase in F-scores as the size of the model increases. This is to be expected, since the criterion for adding a template group is increased accuracy on the development set. However we observe overfitting in the final iteration, as scores fall for all sets. This is due to the fact that the feature selection was run on FTB$\alpha \leq 20$ data and therefore overfits when the templates are used on more complete data. The continued increase on the FTB$\alpha \leq 20$ development set is therefore not a good stopping criteria for the feature selection. Instead, we propose that an increasing F-score on the full FTB$\alpha$ development set be the continuation criterion for feature selection. In this case, the best model produced is the 34-template model at iteration 5 with an F-score of 83.50 for FTB$\alpha$ and 85.95 for sentences of 40 tokens or fewer.

**Templates in the model.** The back-off style construction of template groups means that each template group is identifiable by its largest template (the others being subsumed by this template in terms of the conditions they contain). We therefore summarise the templates added to the final model in Table 6. The full template list can be found in Table 14 in Appendix B.

| Iteration | Templates added to final model |
|---|---|
| 1 | q0(word) & s0(t,c,_) & s1(t,c,_) |
| 2 | q0(tag) & s0(l,h,tag) & s0(l,h,sword) |
| 3 | q1(tag) & s0(r,h,sword) & s1(l,h,tag) |
| 4 | q0(sword) & s0(r,h,tag) & s1(t,h,word) |
| 5 | s0(t,c,_) & s0(r,c,_) & s1(r,h,word) |

Table 6: Method A1: A summary of template groups added at each iteration, as defined by the largest template in each group.

The template group with the highest accuracy on the development set is the group containing a combination of the conditions ‘q0(word)’, ‘s0(t,c,_)’ and ‘s1(t,c,_)’, which correspond to the word of the first queue element and the constituent types of the top of the first two stack elements. Also deemed important and so added early on (in the second iteration) are templates containing a combination of the POS tag of the first queue element and the POS tag and smoothed word of the head of the left child of the top stack element. We can see that the templates use elements near to the top of the stack and the beginning of the queue, not going further than two elements on the queue and two elements on the stack.

### 4.3.2 Method B1 (Boosting with template grouping)

**Heuristic: Template grouping.** The heuristic used here is the same back-off style template grouping as used for the method A1.

**Feature selection.** The principal advantage of the boosting method is its increased efficiency over the standard forward wrapper method. Method B1 is very similar to method A1 in that the search space is the same and therefore the same number of models need to be trained to achieve a model of approximately the same size. The difference lies in the speed of training. With method A1, the model trained at each round is a model that increases in size at each iteration and therefore takes longer to converge in training. With method B1, the models trained remain roughly the same size (with slight fluctuations in the size of template groups as the feature selection progresses) and therefore convergence can be guaranteed at a lower number of perceptron iterations. We choose to fix a very low number of perceptron iterations to reduce the risk of overfitting. We fix the number of perceptrons at 2, which could be considered underfitting, but appears to be sufficient in this case to make distinctions between the informativeness of features, whilst also decreasing the time necessary for feature selection. Run on FTB$\alpha \leq 20$, the progression of the feature selection is shown in Table 7.

| Iter | Tpls added | Time (hrs) | Total time (hrs) |
|---|---|---|---|
| 1 | 7 | 6.88 | 6.88 |
| 2 | 6 | 6.93 | 13.81 |
| 3 | 7 | 5.54 | 19.35 |
| 4 | 5 | 6.16 | 25.52 |
| 5 | 6 | 6.18 | 31.69 |
| 6 | 6 | 5.25 | 36.94 |
| 7 | 3 | 5.63 | 42.57 |
| 8 | 5 | 5.32 | 47.89 |

Table 7: Model characteristics for method B1 at each round of the feature selector.

In terms of efficiency, method B1 is much faster than A1, with a slightly decreasing time taken at each round, because the search space decreases by one template group each time. Due to the fact a constant number of perceptron iterations are done at each round (and also fewer perceptron iterations in general than for the method A1, the times in general are greatly reduced, although still in the order of several hours per iteration.

**Accuracy on the development set.** We retrained the templates obtained at each round of the selector using a global optimisation on FTB$\alpha$, as for method A1. The resulting F-scores are shown in Table 8. We observe a general, although variable increase in F-scores up to the 5th round of the feature selector, the highest scores being observed for the model of size 31, with an F-score of 81.72 for FTB$\alpha$ and 80.43 for sentences of 40 tokens or fewer. After the 5th round we see a slight decrease in scores, probably a sign of overfitting.

| | | F-score with punctuation | | | F-score without punctuation | | |
|---|---|---|---|---|---|---|---|
| Iter | Size | FTB$\alpha$ $\leq 20$ | FTB$\leq 40$ | FTB$\alpha$ | FTB$\alpha$ $\leq 20$ | FTB$\leq 40$ | FTB$\alpha$ |
| 1 | 7 | 84.88 | 77.88 | 73.66 | 85.93 | 79.16 | 75.80 |
| 2 | 13 | 87.73 | 83.00 | 79.33 | 88.85 | 84.27 | 81.07 |
| 3 | 20 | 87.78 | 82.90 | 79.88 | 88.94 | 84.17 | 81.65 |
| 4 | 25 | 87.43 | 82.94 | 79.78 | 88.59 | 84.25 | 81.62 |
| 5 | 31 | 87.94 | **83.10** | **80.05** | 88.99 | **84.39** | **81.72** |
| 6 | 37 | 87.47 | 82.65 | 79.74 | 88.51 | 83.92 | 81.42 |
| 7 | 40 | 87.70 | 82.87 | 79.97 | 88.71 | 84.10 | 81.67 |
| 8 | 45 | **88.11** | 82.98 | 79.83 | **89.33** | 84.30 | 81.58 |
| Method A1 (31 templates) | | | | | | 85.95 | 83.50 |

Table 8: Method B1: F-score on the FTB$\alpha$ development set, shown after each round of the feature selector

What is interesting is the very sharp increase in F-score between the first and second iterations and the very gradual increase thereof. With only 13 templates, the model scores 81.07 on FTB$\alpha$, which is higher than the score of 80.57 scored by the method A1 with a model of 14 templates. However the behaviour of the boosting algorithm is unexpected, since we would expect accuracies to increase over a larger number of iterations.

**Templates in the model.** As with method A1, we can summarise the templates in the model by listing the tri-conditional templates added at each round of the feature selector, as shown in Table 9. We indicate the templates also selected by method A1 in bold red. The full template list for this model can be found in Appendix B.2.

### 4.3.3 Method B2 (Boosting with an additive search space)

**Heuristic: An additive search space.** The heuristic used here approaches the principle of template subsumption from the other direction to template grouping. The idea is to start with a very small search space containing only uni-conditional templates, which expands to accept bi-conditional and tri-conditional templates as new templates are added. The approach should in principal favour the addition of smaller templates to the final fit, which could reduce the total number of calculations made. Templates are added to the search space once the templates containing their conditions have been chosen to be added. For example 'T[3] = q0.word & q1.word' is added once the uni-conditional templates 'T[6] = q0.word' and 'T[7] = q1.word' have been added to the fit. The template 'T[2] = q0.word & q1.word & q2.word' can be added only once all three conditions are included in templates that have been added to the fit. This second heuristic is more extreme than template

| Iteration | Templates added to final model |
|---|---|
| 1 | **q0(word) & s0(t,c,_) & s1(t,c,_)** |
| 2 | q0(word) & s0(t,h,word) & s1(l,h,word) |
| 3 | q1(sword) & s0(r,h,word) & s1(t,h,word) |
| 4 | **q0(sword) & s0(r,h,tag) & s1(t,h,word)** |
| 5 | q0(sword) & s0(t,c,_) & s1(t,h,word) |
| 6 | q0(word) & s0(l,h,word) & s1(r,h,word) |
| 7 | q1(word) & s0(r,h,word) & s1(t,h,sword) |
| 8 | q0(sword) & s0(t,c,_) & s1(l,h,word) |
| 9 | q0(word) & s0(r,h,word) & s1(l,h,sword) |

Table 9: Method B1: A summary of template groups added at each iteration, as defined by the largest template in each group. Those that were also added by method A1 are shown in bold and in red.

grouping in terms of reducing the search space, as it reduces the initial search space to one that is the same size as the number of possible conditions. It also imposes an order on the templates added to the model. However it does not suppose that templates must be added in batches and therefore potentially more refinement can be made to final models as basic units are feature templates and not template groups.

**Feature selection.** This method proceeds template by template, selecting the template with the best weighted accuracy at each round. Each weak model is very small and therefore the perceptron tends to converge in a single round. We ran the feature selector on FTB$\alpha \leq 20$ (for a comparison with the previous two methods) and on FTB$\beta$, which contains more morphological information. The time taken to select a feature is very short (approximately 50 seconds when run on FTB$\alpha \leq 20$ and 10 minutes when run on FTB$\beta$) and therefore the feature selector can be run a larger number of times, even on the more complete dataset. The feature selector was stopped manually after approximately 100 iterations when trained on FTB$\alpha \leq 20$ data and stopped automatically after 37 iterations when trained on FTB$\beta$.

**Accuracy on the development set.** We retrained the templates produced on the FTB$\alpha$ dataset (for the feature selection run on FTB$\alpha \leq 20$) and on the FTB$\beta$ dataset (for the feature selection run on French SPMRL). Results are shown in Tables 10 and 11.

The results are very poor for the feature selection run on the FTB$\alpha \leq 20$ dataset, the best model for the entire dataset being around 30 templates with an F-score almost 10 points below the best score achieved by the method B1 and over 11 points below method A1. It is telling that the best model for parsing FTB$\alpha \leq 20$ is the largest model whose results are shown here (at 50 templates), suggesting that the templates selected are more adapted to parsing shorter sentences. The best model for the complete data is one which contains 50 templates, but the scores, at 73.47 for FTB$\alpha$ and 77.24 for sentences of 40 tokens or

| | F-score with punctuation | | | F-score without punctuation | | |
|---|---|---|---|---|---|---|
| Size | FTB$\alpha \leq 20$ | FTB$\leq 40$ | FTB$\alpha$ | FTB$\alpha \leq 20$ | FTB$\leq 40$ | FTB$\alpha$ |
| 10 | **82.18** | 74.96 | 71.06 | **83.27** | 76.24 | 72.73 |
| 20 | 81.59 | 74.65 | 70.98 | 81.59 | 76.03 | 72.70 |
| 30 | 81.23 | 74.92 | 71.18 | 81.23 | 76.38 | 72.89 |
| 40 | 81.63 | 74.74 | 70.94 | 81.63 | 76.19 | 72.72 |
| 50 | 81.33 | **75.26** | **71.77** | 81.33 | **77.24** | **73.47** |
| Method A1 (34 templates) | | | | | 85.95 | 83.50 |
| Method B1 (31 templates) | | | | | 80.93 | 81.72 |

Table 10: Method B2: F-score on FTB$\alpha$, shown after each 10 rounds of the feature selector

| | F-score with punctuation | | F-score without punctuation | |
|---|---|---|---|---|
| Size | FTB$\beta \leq 40$ | FTB$\beta$ | FTB$\beta \leq 40$ | FTB$\beta$ |
| 5 | 77.26 | 72.92 | 78.56 | 74.64 |
| 10 | 79.55 | 75.82 | 80.89 | 77.52 |
| 15 | 79.55 | 75.97 | 80.91 | 77.80 |
| 20 | 80.21 | 76.40 | 81.66 | 78.17 |
| 25 | 80.55 | 77.24 | 82.08 | 79.18 |
| 30 | 81.34 | 77.78 | 82.81 | 79.71 |
| 37 | **82.01** | **78.47** | **83.41** | **80.21** |
| Method A1 (34 templates) | | | 85.95 | 83.50 |
| Method B1 (31 templates) | | | 80.93 | 81.72 |

Table 11: Method B2: F-score on FTB$\beta$, shown after approximately 5 rounds each time.

fewer, are far inferior to the scores obtained for the preceding two methods. The scores appear erratic as the model grows in size, suggesting that the feature selection was not very successful in this case.

The same method, when run on the complete dataset (FTB$\beta$), produces more encouraging results. The best model is the one containing all 37 templates, with an F-score of 80.21 for all data and 83.41 for sentences of 40 tokens or fewer. We can hypothesise that the boosting method functions better when run on complete data. This is supported by the fact that the performance on the FTB$\alpha \leq 20$ dataset is similar to that of the model run on the smaller dataset, but the model obtained from feature selection on complete data produced far superior scores on the full dataset.

**Templates in the model.** Since the templates are not added by groups and are therefore not easily summarisable, we choose not to list the templates here in the best model. The complete model can be found in Appendix B.3. Here we will simply evoke the characteristics of the templates added. In total there were 8 uni-conditional, 10 bi-conditional and 18 tri-conditional templates, which is far more tri-conditional templates than with the heuristic by template grouping. The only lexical values instantiated were related to the word (i.e.

no morphological information was used in the model). Three elements of the queue were used and two elements from the stack. The constituent labels of the stack were used for the top element of the first and second stack elements and for the left daughter of the first stack element.

It is surprising that no extra morphological information was used in the chosen templates. This could either be linked to the slightly strange behaviour of the boosting algorithm, evoked briefly in the results of method B1 and discussed in more detail in Section 4.5, or to the fact that individually, the lexical information is too weak to be selected. It could be that a combination of these morphological values, and therefore a grouping of templates, is necessary to take into account their ability to co-predict[1]. This was not possible with this method due to the fact that additive forward selection does not function well for co-predictors and especially that the heuristic used does not allow the addition of multi-conditional templates before the uni-conditional templates they subsume are added.

## 4.4 Analysis

### 4.4.1 Model performance during the feature selection

Given that method B2 did not select any extra morphological values, despite the fact that it was trained on the FTB$\beta$ dataset, both B2 results can be compared against those obtained from methods A1 and B1. Figure 22 shows the progression of the F-scores on the development set for the models constructed with each method A1, B1 and B2. Two different models are shown for method B2, one for which feature selection was performed on the FTB$\alpha \leq 20$ dataset and the other for which it was performed on the FTB$\beta$ dataset.

### 4.4.2 Generalisation performances

The performances of the best model for each method, tested on FTB$\alpha$ and FTBFTB$\beta$ test sets, are shown in Table 12 and compared with results from two state-of-the-art models, the Berkeley parser (Petrov *et al.*, 2006) and the manually chosen model presented in (Crabbé, 2014). The method A1 achieves the best score at 2 points higher than the manually selected model. The boosting method B2, run on the FTB$\beta$ data, produces a slightly lower score, equalling the score obtained by the Berkeley parser.

### 4.4.3 Efficiency of feature selection

Although the boosting method achieves slightly lower scores, it shows a huge increase in efficiency as shown by the cumulative selection times shown in Figure 23. Whilst method

[1] Co-predictors are units that do not necessary perform well individually but when combined are high performers.

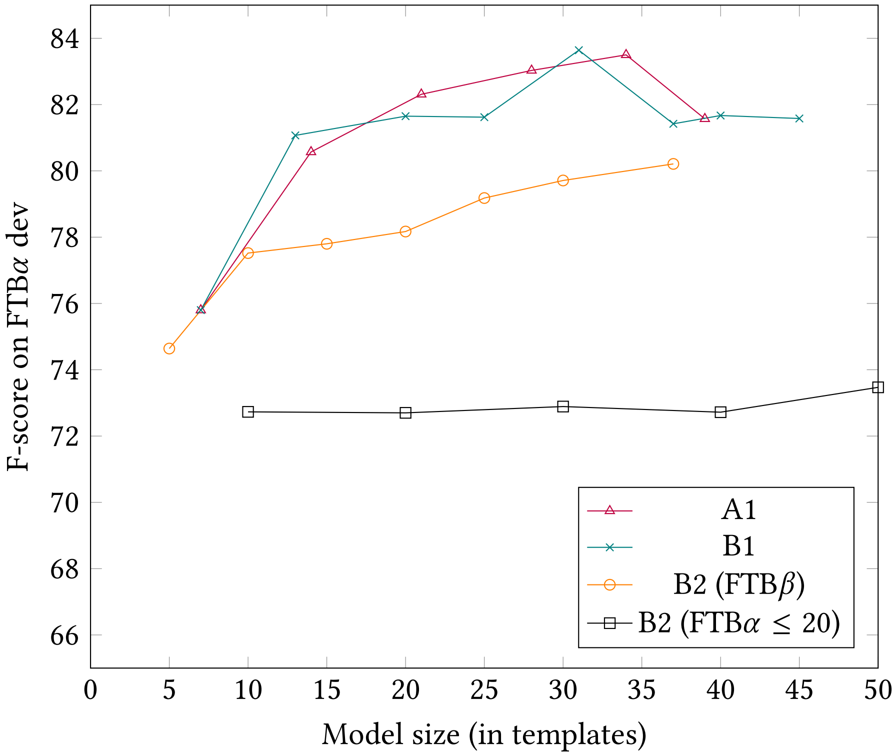


Figure 22: The F-scores for the development set for each method, shown relative to the size of the model selected

| Parser | Run on | F (FTB≤40) | F (FTB$\alpha$) | Rank |
|---|---|---|---|---|
| Berkeley (Petrov *et al.*, 2006) | - | 83.16 | 80.73 | 4 |
| Manual model (Crabbé, 2014) | - | 84.33 | 81.43 | 3 |
| **Method A1 (34 templates)** | FTB$\alpha \le 20$ | **86.27** | **83.43** | 1 |
| Method B1 (31 templates) | FTB$\alpha \le 20$ | 84.78 | 81.81 | 2 |
| Method B2 (50 templates) | FTB$\alpha \le 20$ | 75.16 | 71.76 | 6 |
| Method B2 (37 templates) | FTB$\beta$ | 83.84 | 80.73 | 4 |

Table 12: A comparison of model scores on FTB data using the best models produced by each method. Scores are given without taking into account punctuation.

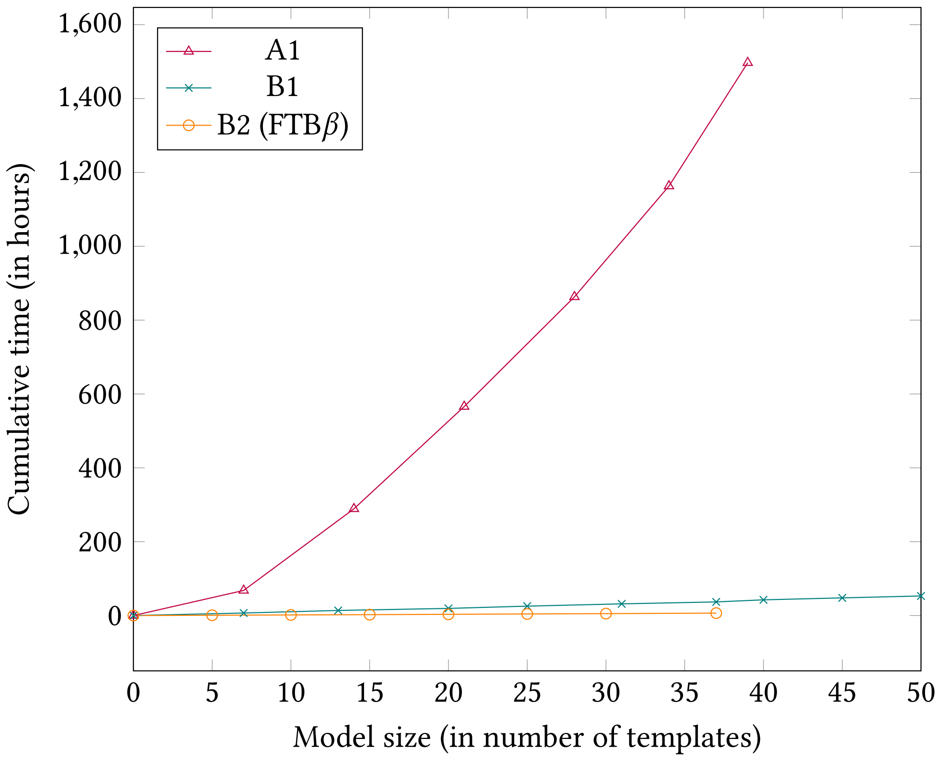


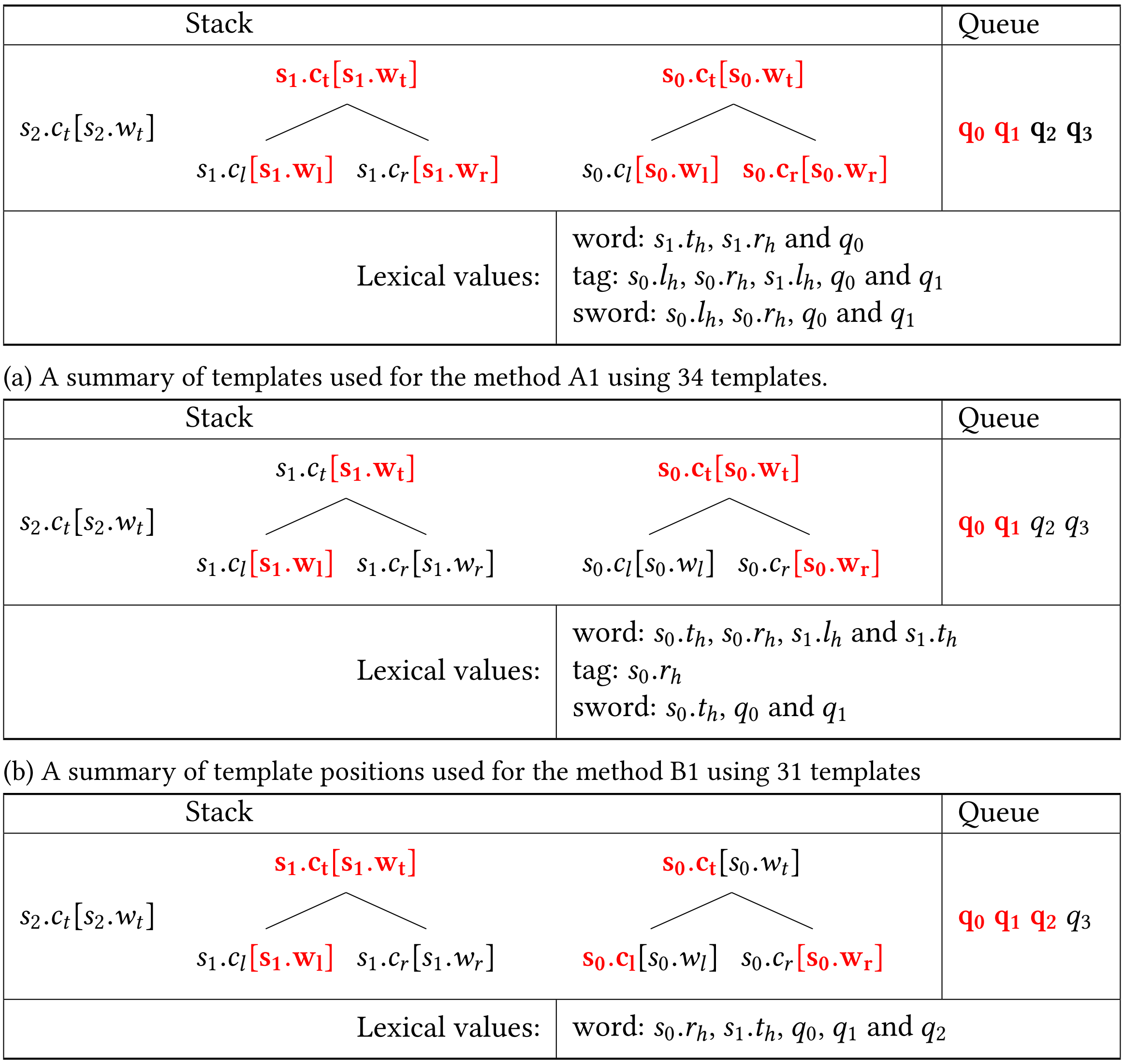
Figure 23: The cumulative time taken for each feature selection method, shown against the increasing size of the model

A1 shows an increasing selection time at each round, the boosting method (both B1 and B2) demonstrate a quasi-constant selection time at each round, B2 being the most feasible method for large, high-dimensional datasets.

### 4.4.4 Descriptive analysis of model templates

To get an insight into what sort of information each model uses, we present here in Figure 24 the positions and lexical values instantiated by the best model for each method, highlighting in red the configuration positions used in the feature templates that make up each model.

We observe that method A1 uses more lexical values and all three types of values available, whereas method B1 uses far fewer. Method B2, trained on different data does not use any other morphological values other than the word. All models use the first two queue elements, the constituent label for the topmost of the top stack element and lexical information relative to the heads of the left daughter of the second topmost stack element and of the right daughter of the topmost stack element. This could indicate the importance of these particular elements in capturing generalisations, although a careful and thorough investigation of why these elements in particular are statistically pertinent is necessary before any real conclusions can be drawn.


Stack
Queue
$s_2.c_t[s_2.w_t]$
$s_1.c_t[s_1.w_t]$
$s_1.c_l[s_1.w_l]$ $s_1.c_r[s_1.w_r]$
$s_0.c_t[s_0.w_t]$
$s_0.c_l[s_0.w_l]$ $s_0.c_r[s_0.w_r]$
$q_0$ $q_1$ $q_2$ $q_3$
Lexical values:
word: $s_1.t_h$, $s_1.r_h$ and $q_0$
tag: $s_0.l_h$, $s_0.r_h$, $s_1.l_h$, $q_0$ and $q_1$
sword: $s_0.l_h$, $s_0.r_h$, $q_0$ and $q_1$

(a) A summary of templates used for the method A1 using 34 templates.

Stack
Queue
$s_2.c_t[s_2.w_t]$
$s_1.c_t[s_1.w_t]$
$s_1.c_l[s_1.w_l]$ $s_1.c_r[s_1.w_r]$
$s_0.c_t[s_0.w_t]$
$s_0.c_l[s_0.w_l]$ $s_0.c_r[s_0.w_r]$
$q_0$ $q_1$ $q_2$ $q_3$
Lexical values:
word: $s_0.t_h$, $s_0.r_h$, $s_1.l_h$ and $s_1.t_h$
tag: $s_0.r_h$
sword: $s_0.t_h$, $q_0$ and $q_1$

(b) A summary of template positions used for the method B1 using 31 templates

Stack
Queue
$s_2.c_t[s_2.w_t]$
$s_1.c_t[s_1.w_t]$
$s_1.c_l[s_1.w_l]$ $s_1.c_r[s_1.w_r]$
$s_0.c_t[s_0.w_t]$
$s_0.c_l[s_0.w_l]$ $s_0.c_r[s_0.w_r]$
$q_0$ $q_1$ $q_2$ $q_3$
Lexical values:
word: $s_0.r_h$, $s_1.t_h$, $q_0$, $q_1$ and $q_2$

(c) A summary of template positions used for the method B2 (FTB$\beta$) using 37 templates

Figure 24: A comparison of configuration elements used in the models resulting from each method

## 4.5 Analysis of the boosting approach

The boosting approach in its current state produces slightly lower scores than the standard forward wrapper, even when run on more complete data. It is clearly advantageous in terms of its efficiency, which means that it can be realistically used on very high-dimensional data. However the behaviour of the boosting algorithm is not quite as expected. Although we observe a general increase in generalisation accuracy in the model produced at each round of the feature selector (until overfitting occurs), the exponential loss at each round does not decrease as expected. For example, Table 13 illustrates the full details of the feature selector for method B1 and we can see that the loss function does not decrease at each round.

| Iter | Loss | Templates added | Time (hrs) | Total time (hrs) |
|---|---|---|---|---|
| 1 | 161394 | 7 | 6.88 | 6.88 |
| 2 | 161527 | 6 | 6.93 | 13.81 |
| 3 | 161512 | 7 | 5.54 | 19.35 |
| 4 | 161408 | 5 | 6.16 | 25.52 |
| 5 | 161420 | 6 | 6.18 | 31.69 |
| 6 | 161433 | 6 | 5.25 | 36.94 |
| 7 | 161395 | 3 | 5.63 | 42.57 |
| 8 | 161401 | 5 | 5.32 | 47.89 |
| 9 | 161402 | 5 | 5.13 | 53.02 |

Table 13: Model characteristics and results for method B1 at each round of the feature selector.

Figure 25, a plot of the boosted prediction accuracy for both training and development sets, shows that it is erratic in both cases and even appears to globally decrease. We observe that the boosted prediction accuracy (the weighted prediction of all weak learners added to the final model, tested on the training and development sets) is not indicative of the final score of the model once it has been retrained on global data. However we would expect to see a general trend of increasing accuracy over time. The second iteration often displays a decrease in boosted prediction accuracy. The reason for this is that if the second learner added has a higher coefficient than the first (because it is based on weighted error), its prediction will dominate over the first. Since the first weak learner added is necessarily the superior one in terms of prediction, as it was selected first, the boosted accuracy will necessarily be lower in the second round, since it will be entirely based on the predictions of the second model. In later rounds we see that boosted accuracy appears to decrease and to converge to around 35%, which is incidentally the percentage of actions that belong to the most common class of action ‘shift’. A look at the individual predictions made by each of the weak classifiers for individual examples reveals that a majority of models predict ‘shift’. This means that after a certain number of these classifiers are added, the action ‘shift’ will almost always receive the highest weighted vote and the boosted accuracy is bound to converge to the same percentage as the percentage of ‘shifts’ in the corpus. We would nevertheless expect the boosting algorithm to succeed in choosing weak classifiers to target particular lacunas and not to concentrate on the most common action.

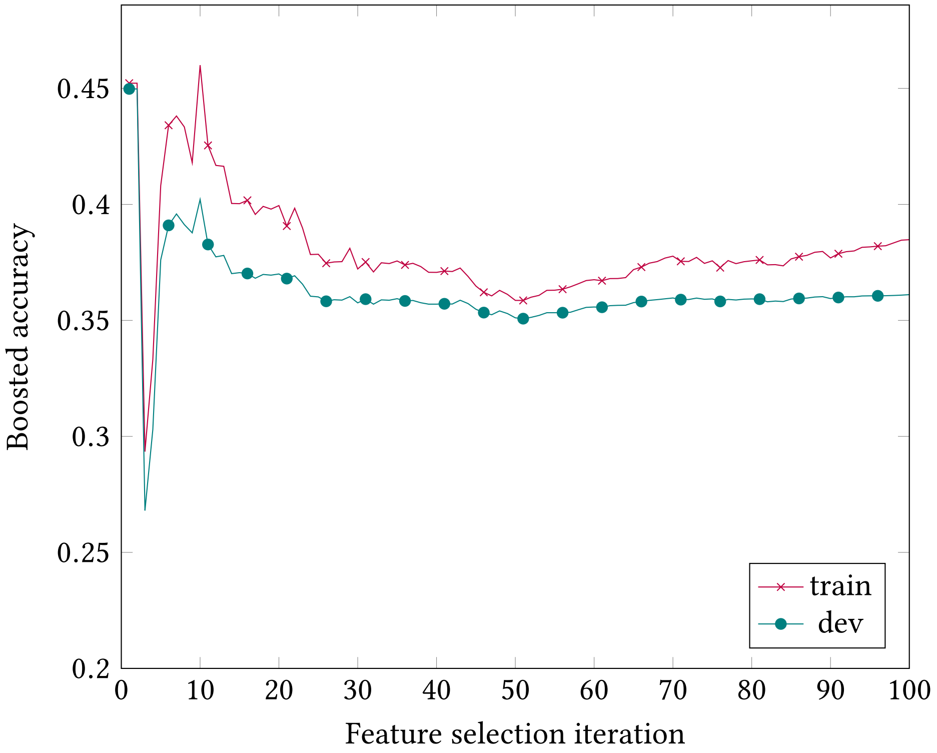


Figure 25: The boosted prediction accuracies for the FTB$\beta$ train and development sets for method B2.

We have several possible explanations for this strange behaviour and hope that further investigations in later work will provide a solution to the problem, improving the results of the feature selector in the meantime. The first possibility is an implementation problem, which must be considered alongside other hypotheses. However there could also be more theoretical explanations, namely the fact that feature selection is done using local optimisation, the properties of the dataset and the properties of the mini-classifiers added at each step.

Many of these ideas can be linked to the insufficient capacity of the weak learners to ensure AdaBoost's guarantees, and further work needs to be done to define what properties these weak learners must have to do so.

**SAMME's guarantees.** Although Zhu *et al.*'s (2006) multi-class algorithm (SAMME) has theoretical guarantees as long as the weak classifiers have a weighted error above random error ($\frac{1}{K}$), in practice, this condition appears to be too weak, a point that is also mentioned in Mukherjee and Schapire (2011). It appears that the algorithm is not always guaranteed to drive down the training error, especially if the weak classifiers have an error very close to random. In their article, Zhu *et al.* demonstrate their algorithm on a three-class simulation sample, whereas we have a situation in which there are over a hundred classes. They run the feature selector for 600 rounds, far longer than our experiments here, which could explain why we do not see a general trend as they do. Their experiments show a very spiked curve representing the decreasing test error, showing that the test error does not always decrease between successive rounds, something that we too observe. It is possible that the threshold they mention is theoretically sound but insufficient for a practical application, unless the algorithm is run a very large number of times. In our case, it is important not to have to run the feature selector a large number of times because we prefer model compactness. We therefore need to choose an empirically acceptable threshold, which we can be sure will guarantee the training loss to decrease, even when run for a relatively low number of rounds and with a high number of classes. This can be made possible by examining the properties of the data to find the true random error weight, given the noisiness of the data and the class imbalances.

### 4.5.1 Properties of the weak learners.

Aside from the threshold for defining a weak learner, which appears not to be sufficiently restrictive, we can also examine the properties of our weak learners to decide why they are too weak in the first place. The point is demonstrated in particular by the fact that method B2, run on FTB$\beta$, stopped naturally after 37 rounds. This indicates that there were no more templates that had a high enough accuracy after the 37th round according to the threshold set by Zhu *et al.*.

**Strength of weak learners.** Individual templates appear to be too weak for boosting to work as expected. We would have expected the method B2 to have produced a better model than method B1, given that the basic units are feature templates rather than template groups and there is a potentially a greater capacity to fine-tune the model. However what we observe is the opposite: the best model produced by B2 is one with 37 templates and a lower score than the best model produced by B1 with only 31 templates. This suggests that grouping templates into mini-parsers that act as weak learners functions better than using templates individually as with the additive search space heuristic.

**Co-predictors.** The additive search space heuristic also poses a problem for co-predictors, since templates cannot be added in groups and therefore only individually high-performing templates can ever be added to the model. Half of the templates added to the best model produced by method B2 were tri-conditional templates and the method appeared to add the multi-conditional templates as soon as they were made available. This suggests that these multi-conditional templates were almost always individually stronger than ones with fewer conditions. This also provides a plausible explanation for the lack of morphology in the templates added. If each template using a morphological attribute is individually too weak to be added to the model, the multi-conditional templates containing these morphological attributes, which stand a better chance of being strong enough, are never added to the search space, according to the heuristic defined. What is more, the individual templates can never be added in batches because templates are added one by one.

**Avoid overfitting.** Another observation is the fact that in both methods B1 and B2, the results appear better when weak learners are trained using a very small number of perceptron iterations. We observe that the learners overfit very rapidly, especially in method B2 where the weak learners are individual templates, and a more thorough work into determining the optimal number of iterations is necessary.

### 4.5.2 Conclusion and perspectives

There are therefore several paths that need to be explored to improve the method by boosting, which is already very promising in terms of efficiency and produces reasonably high scores when tested on French data. Two hyperparameters need to be investigated: i) the empirical threshold needed to ensure proper functioning of the multi-class AdaBoost algorithm and ii) the effect the number of perceptron iterations performed during training has on the feature selection. This is especially linked to the fact that the perceptron is known to overfit if run for too many iterations. However to ensure that weak learners are strong enough to perform higher than the more constraining threshold that will be fixed, and to be able to take into account co-predictors, the heuristics will have to be refined. Whilst template grouping appears to perform better than the additive search space heuristic, it does not reduce the search space enough to be able to perform an efficient feature selection, especially on data containing more morphological attributes. It is therefore not possible to use this heuristic to test our hypothesis on more morphologically rich languages than French. However the additive search space heuristic is far too constraining in that individual templates are too weak to be selected, and the method fails to take into account co-predictors. We propose to further study these heuristic methods to produce one in which templates are grouped to allow for co-predictors and to ensure sufficiently strong weak learners. However the method will have to regroup templates differently from the back-off style grouping presented here in order to reduce the search space to a greater extent. One option is to rely on the order of elements in the stack and the queue to first use templates closest to the focus tokens before adding templates that use more distant elements, as in Nilsson and Nugues (2010). Another option is to perform an initial fit of the templates to establish an order between them, which could be used to group the templates and to guide the search of the feature space. Finally, we propose to develop a way of integrating default templates into the initial model, as in Ballesteros and Nivre (2014) in order to accelerate the initial steps of the feature selection process. This step can also be considered a way of integrating linguistic intuitions into the feature selection process. Although linguistic intuitions are a too complex way of accurately selecting an optimal subset of individual templates, they are in some cases sufficient for identifying general trends of positions and of values that might be useful. If this information is provided, it can be used to limit the feature space and to guide the feature selection process.

# CHAPTER 5

# CONCLUSIONS AND FUTURE WORK

We have proposed a new approach to feature selection for syntactic parsing, designed to handle lexically rich (and therefore high dimensional) data. We have seen that using boosting for feature selection has the advantage over state-of-the-art methods of being very efficient, taking a constant time at each iteration of the feature selector. It therefore has the merit of being more generalisable to larger and more complex datasets, without falling into the trap of the curse of dimensionality.

The results on French data show that the method can currently produce comparable results to carefully selected manual models, such as the one presented in Crabbé (2014). We hope to improve on these scores in later work by testing the practical limits of the SAMME algorithm, which currently appear to be too weak in our case, and by adjusting the heuristics used to define the feature space to ensure weak learners are sufficiently strong. In terms of morphology, we have not yet been able to test our hypothesis that added lexical information can aid parsing, due to the fact that our method is currently a poor selector of co-predictors and the heuristic used on the complete data is too constraining for these particular attributes. The two heuristics, template grouping and an additive search space, will need to be replaced by a method capable of reducing the feature space, but without imposing the same strong constraints as the additive search space heuristic. We will use the results of our experiments and inspiration from heuristic methods used in the literature to design an adapted heuristic method. We also envisage the possibility of re-integrating linguistic intuitions into this heuristic, by allowing the integration of underspecified template classes to guide the search.

Having achieved promising (although clearly improvable) results on French data, we plan to test our method on typologically varied data, for which more morphological information is provided. We are confident that our approach, unlike alternative approaches, is capable of handling this high-dimensionality, and we also hope to test our hypothesis that using extra morphological information can improve parsing performances for these languages.

# APPENDIX A ADABOOST

## A.1 Explaining the use of Adaboost's exponential loss function (two-class)

Finding the base function and its coefficient that minimise the loss can be performed in two steps, by first solving the function for $g^{(T)}$ and then by using the formula obtained to calculate the optimal value for $\alpha$.

To calculate the base function that minimises the exponential loss, let us revisit Equation 3.8.

$$\left(\alpha^{(T)}, g^{(T)}\right) = \underset{(\alpha,g)}{\operatorname{argmin}} \sum_{i=1}^{D} w_i^{(T)} \exp\left(-y_i\ \alpha\ g^{(T)}(x_i)\right) \tag{3.8 abridged}$$

For any value of $\alpha > 0$, $g^{(T)}$ can be expressed as follows:

$$g^{(T)} = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{D} w_i^{(T)} \exp\left(-y_i\ \alpha\ g(x_i)\right) \tag{A.1}$$

Since this is the two-class case, the value of $y$ (and therefore $g$) is in $\{1, -1\}$ and so we can separate the examples that were correctly classified from those that were incorrectly classified and redistribute the equation as follows:

$$= \underset{g}{\operatorname{argmin}} \quad e^{\alpha} \sum_{y_i \neq g(x_i)} w_i^{(T)} \ + \ e^{-\alpha} \sum_{y_i = g(x_i)} w_i^{(T)} \tag{A.2}$$

$$\begin{aligned} &= \underset{g}{\operatorname{argmin}} \quad e^{\alpha} \sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i)) \ + \ e^{-\alpha} \sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i = g(x_i)) \\ &= \underset{g}{\operatorname{argmin}} \quad e^{\alpha} \sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i)) \ + \ e^{-\alpha} \sum_{i=1}^{D} w_i^{(T)} \ - \ e^{\alpha} \sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i)) \end{aligned} \tag{A.3}$$

$$= \underset{g}{\operatorname{argmin}} \quad \left(e^{\alpha} - e^{-\alpha}\right) \sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i)) \; + \; e^{-\alpha} \sum_{i=1}^{D} w_i^{(T)} \tag{A.4}$$

By reducing the terms in this expression that do not effect the minimum $g$, we can see that the chosen base function that minimises the loss is the one that has the lowest weighted error on the training set, as shown in Equation A.5

$$g^{(T)}(x) = \underset{g}{\operatorname{argmin}} \sum_{i=1}^{D} w_i^{(T)} \; \mathbb{I}\Big(y_i \neq g(x_i)\Big), \tag{A.5}$$

This last expression obtained can now be used in Equation A.4 to calculate the coefficient that minimises the loss. We can divide by a constant factor $\sum_{i=1}^{D} w_i^{(T)}$ to simplify the formulation, without altering the minimum value:

$$\Big(\alpha^{(T)}, g^{(T)}(x_i)\Big) = \underset{g}{\operatorname{argmin}} \quad \left(e^{\alpha} - e^{-\alpha}\right) \frac{\sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i))}{\sum_{i=1}^{D} w_i^{(T)}} \; + \; e^{-\alpha}, \tag{A.6}$$

and if we consider $\frac{\sum_{i=1}^{D} w_i^{(T)} \mathbb{I}(y_i \neq g(x_i))}{\sum_{i=1}^{D} w_i^{(T)}}$ to be the weighted error $err$ on the training data, this can be simplified as follows:

$$\Big(\alpha^{(T)}, g^{(T)}(x_i)\Big) = \underset{g}{\operatorname{argmin}} \quad \left(e^{\alpha} - e^{-\alpha}\right) \cdot err \; + \; e^{-\alpha}, \tag{A.7}$$

The coefficient $\alpha$ that minimises the loss can now be calculated by deriving this function $f : \alpha \mapsto \left(e^{\alpha} - e^{-\alpha}\right) \cdot err + e^{-\alpha}$ as illustrated in Equation A.8.

$$\begin{aligned} \frac{\mathrm{d}f}{\mathrm{d}\alpha} &= \frac{\mathrm{d}}{\mathrm{d}\alpha} \left(e^{\alpha} - e^{-\alpha}\right) err \; + \; e^{-\alpha} \\ &= e^{\alpha} err + e^{-\alpha} err - e^{-\alpha} \\ &= e^{-\alpha} e^{2\alpha} err + e^{-\alpha} err - e^{-\alpha} \\ &= e^{-\alpha} \left(e^{2\alpha} err + err - 1\right) \end{aligned} \tag{A.8}$$

Let us now seek the value of $\alpha$ for which the following equality stands:

$$\begin{aligned} e^{-\alpha} \left(e^{2\alpha} err + err - 1\right) &= 0 \\ e^{2\alpha} err + err - 1 &= 0 \\ e^{2\alpha} err &= 1 - err \\ e^{2\alpha} &= \frac{1 - err}{err} \\ \alpha &= \frac{1}{2} \log \frac{1 - err}{err} \end{aligned} \tag{A.9}$$

# APPENDIX B FINAL MODELS

## B.1 Method A1

| Iteration | Templates added to final model |
|---|---|
| 1 | q0(word) |
| | q0(word) & s0(t,c,_) |
| | q0(word) & s0(t,c,_) & s1(t,c,_) |
| | q0(word) & s1(t,c,_) |
| | s0(t,c,_) |
| | s0(t,c,_) & s1(t,c,_) |
| | s1(t,c,_) |
| 2 | q0(tag) |
| | q0(tag) & s0(l,h,tag) |
| | q0(tag) & s0(l,h,tag) & s0(l,h,sword) |
| | q0(tag) & s0(l,h,sword) |
| | s0(l,h,tag) |
| | s0(l,h,tag) & s0(l,h,sword) |
| | s0(l,h,sword) |
| 3 | q1(tag) |
| | q1(tag) & s0(r,h,sword) |
| | q1(tag) & s0(r,h,sword) & s1(l,h,tag) |
| | q1(tag) & s1(l,h,tag) |
| | s0(r,h,sword) |
| | s0(r,h,sword) & s1(l,h,tag) |
| | s1(l,h,tag) |
| 4 | q0(sword) |
| | q0(sword) & s0(r,h,tag) |
| | q0(sword) & s0(r,h,tag) & s1(t,h,word) |
| | q0(sword) & s1(t,h,word) |
| | s0(r,h,tag) |
| | s0(r,h,tag) & s1(t,h,word) |
| | s1(t,h,word) |
| 5 | s0(t,c,_) & s0(r,c,_) s0(t,c,_) & s0(r,c,_) & s1(r,h,word) s0(t,c,_) & s1(r,h,word) s0(r,c,_) s0(r,c,_) & s1(r,h,word) s1(r,h,word) |

Table 14: Method A1: The templates added after each round of the feature selector. We present only the first five rounds here, as this was the best model produced.

## B.2 Method B1

| Iteration | Templates added to final model |
|---|---|
| 1 | q0(word) |
| | q0(word) & s0(t,c,_) |
| | q0(word) & s0(t,c,_) & s1(t,c,_) |
| | q0(word) & s1(t,c,_) |
| | s0(t,c,_) |
| | s0(t,c,_) & s1(t,c,_) |
| | s1(t,c,_) |
| 2 | q0(word) & s0(t,h,word) |
| | q0(word) & s0(t,h,word) & s1(l,h,word) |
| | q0(word) & s1(l,h,word) |
| | s0(t,h,word) |
| | s0(t,h,word) & s1(l,h,word) |
| | s1(l,h,word) |
| 3 | q1(sword) |
| | q1(sword) & s0(r,h,word) |
| | q1(sword) & s0(r,h,word) & s1(t,h,word) |
| | q1(sword) & s1(t,h,word) |
| | s0(r,h,word) |
| | s0(r,h,word) & s1(t,h,word) |
| | s1(t,h,word) |
| | q0(sword) |
| 4 | q0(sword) & s0(t,c,_) |
| | q0(sword) & s0(t,c,_) & s1(t,h,word) |
| | q0(sword) & s1(t,h,word) |
| | s0(t,c,_) & s1(t,h,word) |
| | q0(word) & s0(l,h,word) |
| | q0(word) & s0(l,h,word) & s1(r,h,word) |
| 5 | q0(word) & s1(r,h,word) |
| | s0(l,h,word) |
| | s0(l,h,word) & s1(r,h,word) |
| | s1(r,h,word) |
| | q1(word) |
| | q1(word) & s0(r,h,word) |
| | q1(word) & s0(r,h,word) & s1(t,h,sword) |

Table 15: Method B1: The templates added after each round of the feature selector.

## B.3 Method B2 - FTB$\beta$

| Iter | Templates added | Iter | Templates added |
|---|---|---|---|
| 1 | s0(t,c,_) | 20 | q1(word) & s0(t,c,_) & s0(r,h,word) |
| 2 | s1(t,c,_) | 21 | s1(t,h,word) |
| 3 | s0(t,c,_) & s1(t,c,_) | 22 | q1(word) & s0(r,h,word) & s1(t,h,word) |
| 4 | q0(word) | 23 | s0(r,h,word) & s1(t,h,word) |
| 5 | q0(word) & s0(t,c,_) & s1(t,c,_) | 24 | q2(word) & s0(r,h,word) & s1(t,h,word) |
| 6 | q0(word) & s1(t,c,_) | 25 | q0(word) & s0(r,h,word) & s1(t,h,word) |
| 7 | q1(word) | 26 | s0(t,c,_) & s0(r,h,word) & s1(t,h,word) |
| 8 | q0(word) & q1(word) & s1(t,c,_) | 27 | s0(t,c,_) & s0(r,h,word) |
| 9 | q1(word) & s0(t,c,_) & s1(t,c,_) | 28 | q0(word) & s1(t,c,_) & s1(t,h,word) |
| 10 | s0(r,h,word) | 29 | s0(t,c,_) & s1(t,c,_) & s1(t,h,word) |
| 11 | q0(word) & s0(r,h,word) & s1(t,c,_) | 30 | q2(word) & s1(t,c,_) |
| 12 | q1(word) & s0(r,h,word) | 31 | s0(t,c,_) & s1(t,h,word) |
| 13 | q1(word) & s0(r,h,word) & s1(t,c,_) | 32 | s0(r,h,word) & s1(t,c,_) |
| 14 | q0(word) & q1(word) & s0(r,h,word) | 33 | q0(word) & s0(t,c,_) |
| 15 | q2(word) | 34 | q1(word) & s1(t,c,_) |
| 16 | q1(word) & q2(word) & s0(r,h,word) | 35 | s0(l,c,_) |
| 17 | q2(word) & s0(t,c,_) & s1(t,c,_) | 36 | s0(t,c,_) & s0(l,c,_) & s1(t,c,_) |
| 18 | s0(t,c,_) & s0(r,h,word) & s1(t,c,_) | 37 | s0(l,c,_) & s1(t,c,_) |
| 19 | q0(word) & q2(word) & s1(t,c,_) | | |

Table 16: Method B2: The templates added after each round of the feature selector.

# BIBLIOGRAPHY

Abeillé, A., Clément, L., and Toussenel, F. (2003). Building a Treebank for French. In *Treebanks*, chapter 10, pages 165–188. Kluwer.

Attardi, G., Dell'Orletta, F., Simi, M., Chanev, A., and Ciaramita, M. (2007). Multilingual Dependency Parsing and Domain Adaptation using DeSR. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 1112–1118.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM Conference. In *IFIP Congress*, pages 125–131.

Ballesteros, M. and Bohnet, B. (2014). Automatic Feature Selection for Agenda-Based Dependency Parsing. In *In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 794–805.

Ballesteros, M. and Nivre, J. (2014). MaltOptimizer: Fast and effective parser optimization. *Natural Language Engineering*, pages 1–27.

Blumer, A., Ehrenfeucht, a., Haussler, D., and Warmuth, M. K. (1989). Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, **36**(4), 929–965.

Bohnet, B. and Kuhn, J. (2012). The Best of Both Worlds – A Graph-based Completion Model for Transition-based Parsers. *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL '12)*, pages 77–87.

Bohnet, B., Nivre, J., Boguslavsky, I., Ginter, F., and Haji, J. (2013). Joint Morphological and Syntactic Analysis for Richly Inflected Languages. *Transactions of the Association for Computational Linguistics*, **1**(2012), 415–428.

Charniak, E. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of the first conference on North American chapter of the Association for Computational Linguistics*, pages 132–139.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, **2**(3).

Collins, M. (1999). *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Collins, M. (2002). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP).*

Collins, M. and Roark, B. (2004). Incremental parsing with the perceptron algorithm. *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics ACL 04*, pages 111–es.

Collins, M. J. (2003). Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, **29**(4), 589–637.

Cortes, C., Kuznetsov, V., and Mohri, M. (2014). Ensemble Methods for Structured Prediction. In *Proceedings of the 31st International Conference on Machine Learning, Beijing, China, 2014. JMLR: W&CP*, volume 32.

Crabbé, B. (2014). An LR-inspired generalized lexicalized phrase structure parser. In *COLING the 25th International Conference on Computational Linguistics*, pages 541–552.

Crabbé, B. (2015). Multilingual discriminative lexicalized phrase structure parsing.

Crabbé, B. and Seddah, D. (2014). Multilingual Discriminative Shift-Reduce Phrase Structure Parsing for the SPMRL 2014 Shared Task. In *First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages.*

Das, S. (2001). Filters, wrappers and a boosting-based hybrid for feature selection. *Engineering*, pages 74–81.

Freund, Y. and Schapire, R. E. (1999). A brief introduction to boosting. *IJCAI International Joint Conference on Artificial Intelligence*, **2**(5), 1401–1406.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Friedman et al 2000 - additive logistic regression - a statistical view of boosting.pdf. *The Annals of Statistics*, **28**(2), 337–407.

Geman, S., Bienenstock, E., and Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, **4**, 1–58.

Gu, Q., Li, Z., and Han, J. (2012). Generalized Fisher Score for Feature Selection. *CoRR*, **abs/1202.3**.

Hall, D., Durrett, G., Klein, D., and Division, C. S. (2014). Less Grammar , More Features. *ACL*, pages 228–237.

Hall, J., Nilsson, J., and Nivre, J. (2007). Single malt or blended? A study in multilingual parser optimization. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing and Conference on Computational Natural Language Learning (EMNLP-CoNLL)*, pages 933–939.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer series in statistics. Springer, New York, 2nd edition.

He, H., Daumé III, H., and Eisner, J. (2013). Dynamic Feature Selection for Dependency Parsing. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1455–1464.

Huang, L. and Sagae, K. (2010). Dynamic programming for linear-time incremental parsing. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1077–1086.

Huang, L., Fayong, S., and Guo, Y. (2012). Structured Perceptron with Inexact Search. *2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technolgies*, pages 142–151.

Johnson, M. (1998). Finite-state Approximation of Constraint-based Grammars using Left-corner Grammar Transforms. In *Proceedings of COLING/ACL*.

Kallmeyer, L. and Maier, W. (2010). Data-Driven Parsing with Probabilistic Linear Context-Free Rewriting Systems. *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 537–545.

Kira, K. and Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. In *AAAI*, pages 129–134.

Klein, D. and Manning, C. (2003). Accurate unlexicalized parsing. In *ACL '03 Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1*, pages 423–430.

Knuth, D. E. (1965). On the translation of languages from left to right. *Information and Control*, **8**, 607–639.

Lei Yu, H. L. (2003). Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the 20th International Conference on Machine Learning (ICML-2003)*, page 856, Washington DC.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, **19**(2), 313–330.

McDonald, R., Pereira, F., Ribarov, K., and Hajič, J. (2005). Non-projective dependency parsing using spanning tree algorithms. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, volume 18, pages 523–530.

Mukherjee, I. and Schapire, R. E. (2011). A theory of multiclass boosting. *Journal of Machine Learning Research*, **14**(1), 437–497.

Müller, T., Schmid, H., and Schütze, H. (2013). Efficient Higher-Order CRFs for Morphological Tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332.

Nilsson, P. and Nugues, P. (2010). Automatic Discovery of Feature Sets for Dependency Parsing. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 824–832.

Nivre, J. (2006). Inductive Dependency Parsing. *Computational Linguistics*, **33**(2), 267–269.

Nivre, J. (2008). Algorithms for Deterministic Incremental Dependency Parsing. *Computational Linguistics*, **34**(4), 513–553.

Nivre, J. and Mcdonald, R. (2008). Integrating Graph-Based and Transition-Based Dependency Parsers. In *Proceedings of ACL-08: HLT*, pages 950–958.

Nivre, J. and Scholz, M. (2004). Deterministic dependency parsing of English text. In *Proceedings of the 20th International conference on Computational Linguistics (COLING 2004)*, volume 4, pages 64–70.

Petrov, S., Barrett, L., Thibaux, R., and Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440.

Sagae, K. and Lavie, A. (2005). A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology - Parsing '05*, pages 125–132.

Sagae, K. and Lavie, A. (2006). Parser combination by reparsing. *Computational Linguistics*, pages 129–132.

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiorkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Wolinski, M., Wroblewska, A., and de la Clergerie, E. V. (2013). Overview of the SPMRL 2013 Shared Task : Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically Rich Languages*, pages 146–182.

Seddah, D., Sandra, K., and Tsarfaty, R. (2014). Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-Rich Languages. In *First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109.

Skut, W., Krenn, B., Brants, T., and Uszkoreit, H. (1997). An Annotation Scheme for Free Word Order Languages. In *Proceedings of the 5th Conference on Applied Natural Language Processing (ANLP)*, page 8, Washington DC.

Tesnière, L. (1959). *Éléments de syntaxe structurale*. Klincksieck, Paris.

Vijay-Shanker, K., Weir, D., and Joshi, A. (1987). Characterizing Structural Descriptions Produced by Various Grammatical Formalisms. In *Proceedings of the 25th Meeting of the Association for Computational Linguistics (ACL 1987)*, pages 104–111.

Viola, P. and Jones, M. J. (2004). Robust Real-Time Face Detection. *International Journal of Computer Vision*, **57**(2), 137–154.

Wang, Q. I., Lin, D., and Schuurmans, D. (2007). Simple training of dependency parsers via structured boosting. *IJCAI International Joint Conference on Artificial Intelligence*, pages 1756–1762.

Wang, X., Lin, X., Yu, D., Tian, H., and Xihong, W. (2006). Chinese Word Segmentation with Maximum Entrop and N-gram Language Model. In *Proceedings of the 5th SIGHAN Workshop on Chinese Language Processing*, pages 138–141.

Zhang, H., Zhang, M., Tan, C. L., and Li, H. (2009). K-Best Combination of Syntactic Parsers. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1552–1560.

Zhang, Y. and Nivre, J. (2011). Transition-based Dependency Parsing with Rich Non-local Features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL '11): shortpapers*, pages 188–193.

Zhu, J., Arbor, A., and Hastie, T. (2006). Multi-class AdaBoost. Technical report, Stanford University.

Zhu, M., Zhang, Y., Chen, W., Zhang, M., and Zhu, J. (2013). Fast and Accurate Shift-Reduce Constituent Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 434–443.

Zhu, Z., Ong, Y.-S., and Dash, M. (2007). Wrapper-Filter Feature Selection Algorithm Using a Memetic Framework. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **37**(1), 1–19.