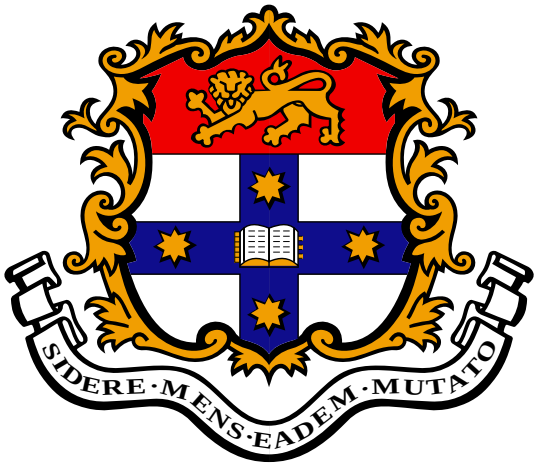
Adaptive Supertagging for Faster Parsing

JONATHAN K. KUMMERFELD
SID: 306150948



Supervisor: James R. Curran

This thesis is submitted in partial fulfillment of
the requirements for the degree of
Bachelor of Science (Honours)

School of Information Technologies
The University of Sydney
Australia

2 November 2009

Abstract

Statistical parsers are crucial for tackling the grand challenges of Natural Language Processing. The most effective approaches to these tasks are data driven, but parsers are too slow to be effectively used on large data sets. State-of-the-art parsers generally cannot process more than one sentence a second, and the fastest cannot process more than fifty sentences a second. The situation is even worse when they are applied outside of the domain of their training data. The fastest systems have two components, a parser, which has time complexity $O(n^3)$ and a supertagger, which has linear time complexity. By shifting work from the parser to the supertagger we dramatically improve speed.

This work demonstrates several major novel ideas that improve parsing efficiency. The core idea is that the tags chosen by the parser are gold standard data for its supertagger. This leads to the second surprising conceptual development, that decreasing tagging accuracy can improve parsing performance. To demonstrate these ideas required extensive development of the C&C supertagger, including implementation of more efficient estimation algorithms and parallelisation of the training process. This was particularly challenging as the C&C supertagger is a state-of-the-art high performance system designed with a focus on speed rather than flexibility.

I was able to significantly improve performance on the standard evaluation corpus by using the parser to generate extremely large new resources for supertagger training. I have also shown that these methods provide significant benefits on another domain, Wikipedia text, without the cost of generating human annotated data sets. These parsing performance gains occur while supertagging accuracy decreases.

Despite extensive use of supertaggers to improve parsing efficiency there has been no comprehensive study of the interaction between a supertagger and a parser. I present the first systematic exploration of the relationship, show the potential benefits of understanding it, and demonstrate a novel algorithm for optimising the parameters that define it.

I have constructed models that process newspaper text 86% faster than previously, and Wikipedia text 30% faster, without any loss in accuracy and without the aid of extra gold standard resources in either domain. This work will lead directly to improvements in a range of Natural Language Processing tasks by enabling the use of far more parsed data.

Acknowledgements

This year has been an epic process that I would not have been able to complete without support from a raft of people. First and foremost, my supervisor, who cracked the whip when necessary and was always there to answer my questions and give me a few more to think about.

A great deal of the implementation work for this project was completed as part of the JHU CLSP Workshop. It was a wonderful opportunity to work in a team for an extended period and I would like to thank the organisers – for making it happen, my supervisor – for bringing our team together, and a fellow honours student, Tim Dawborn – for sharing the products of his culinary expertise. I would also like to thank the entire "Parsing the Web" team, and in particular Jessika Rosener, who I collaborated with on the parallelisation of the training process.

Many of the experiments described in this work required large scale computing resources and would not have been possible without the support of Prof. Harrowell, who gave me access to the Silica computing cluster.

The final form of this thesis is far kinder to the reader, especially those uninitiated to the mysteries of NLP. This would not have been the case without the editing and suggestions provided by many of my friends, in particular Casey Handmer, Anna Katrina Dominguez, Ben Hachey, Ian Oosterhoff, and Nicky Ringland.

Finally, throughout this epic process I have always had the support of my family. Whether they were providing a platform to bounce ideas off, cooking fuel to keep me going, or giving feedback on my work, they have been there for me, and I am extremely grateful.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

Introduction

The aim of this project was to substantially improve the efficiency and accuracy of natural language parsing, without the costly development of more gold standard data. This work falls within the field of Natural Language Processing (NLP), a part of Artificial Intelligence research that focuses on building systems that intelligently use the contents of documents written in natural language.

Parsing is the process of analysing a set of tokens, such as the words in a sentence, and extracting syntactic structure. In some artificial languages, such as computer programming, care has been taken in the design to ensure that each "sentence" has only one possible interpretation. But natural language has evolved over time, and allows ambiguous sentences that are context sensitive. Also the rules (or grammar) that define natural language are not fixed or completely known. This makes parsing extremely difficult.

This ambiguity and lack of a well-defined grammar are challenges for parsers – computer programs designed to extract the syntactic structure of a sentence in natural language. Accurate and efficient parsing is critical in a range of areas. Examples include question answering – to understand the content of documents, computer human interaction – to understand natural language input from users, and machine translation – to ensure the meaning of text is not lost in translation.

The question-answering task involves constructing a system that can understand a question posed in natural language, search for an answer in a collection of documents, and return the answer in an appropriate form (Lehnert, 1977). For example, given the question "When was the last time someone other than the leader of the majority party in the House of Representatives was Prime Minister of Australia?" the system would be expected to give a response such as "1975" or "The 1975 Constitutional Crisis". By providing the syntactic structure of the question, parsers allow us to determine the constraints that define our answer. If we have parsed our document collection we can then determine which facts meet

the constraints in the question. The primary reason for limited use of parsers in this field is that the time required to parse a large document collection is far too large to be feasible.

Another task where parsers are proving useful is automatic translation of text from one language to another. Differences in grammar, idioms, vocabulary and morphology are just a few of the challenges for machine translation systems. Using syntactic information is a popular method of approaching these problems, effectively using syntax to extract meaning and then re-expressing the meaning, rather than trying to translate individual words or phrases (Brown *et al.*, 1990). Parsers are a crucial part of this approach as they provide the syntactic structure of the sentence. The problem with this approach is that the performance of these statistical machine translation systems is highly dependent on the amount of training data used. Faster parsers would enable the exploitation of larger amounts of training data.

A more specific example of one of the applications of parsing is ‘anaphora resolution’, determining which entity a given pronoun refers to (Mitkov, 2002). For example, in the sentence “The theory is that Seymour is the chief designer of the Cray-32, and without him it could not be completed.” the word ‘him’ refers to ‘Seymour’ and the word ‘it’ refers to the ‘Cray-32’. Clearly such understanding is crucial for a system that aims to understand the meaning of documents. People resolve the ambiguity of which pronoun links to which proper noun through a combination of grammatical and general world knowledge, neither of which are as well developed in computer systems as they are in people. Computer systems resolve the ambiguity by considering the syntactic structure of the sentence, as determined by a parser (Ge *et al.*, 1998).

All of these tasks rely on the use of large data sets. The size of these sets is limited by the speed and domain dependence of parsers. At between one and fifty sentences per second, state-of-the-art wide coverage parsers are too slow to be feasibly used. To process the estimated ten trillion words on the English web would take over fifteen thousand years, and the challenge is even greater as more content is constantly being added and large sections, such as Wikipedia, are constantly changing. The number of people using the web and the amount of activity online is continuously increasing, so we cannot rely on increases in computing power to solve this problem. We need algorithmic solutions that improve the efficiency of parsing.

Two possible approaches to improving efficiency are to exploit more training data annotated by humans, or to accept a decrease in accuracy in return for improved speed. Neither of these options are satisfactory,

as the first is extremely costly and domain dependent and the second simply trades speed for accuracy, leaving a new problem.

The big idea that gives the most efficient parsers their speed is 'supertagging' (Bangalore and Joshi, 1999). The parsing process has a time complexity of $O(n^3)$, where n is the length of the sentence. Supertagging, which has a time complexity of $O(n)$, involves reducing the number of possibilities the parser has to consider by assigning a role to each word in the sentence. Using the supertagger to effectively do some of the parsers work leads to a considerable improvement in efficiency, as the parser has a considerably greater time complexity. In this work I take this idea further, passing more work to the supertagger in a novel manner.

1.1 Contributions

In this work I improve the efficiency of a state-of-the-art natural language parser by using the output of the system to retrain one of the earlier stages. The first step in the system is to run the supertagger, which generates tag sets for each word in the sentence. The parser then selects one tag for each word and forms a derivation. The core idea of this thesis is that we can improve efficiency by reducing the set the supertagger supplies for each word to be just the tag that the parser would have used anyway. This novel idea means we can obtain vast amounts of extra "gold standard" training data by using the parser to annotate text. In fact, the resources developed as part of this work were too large for the original architecture of the system.

The original system took several hours to train on forty thousand sentences of annotated data. Even if extra data annotated by people were available, the training process would not have been able to scale up as it would have hit memory constraints and taken far too long. As part of the 2009 Johns Hopkins University Center for Language and Speech Processing Summer Workshop I implemented significant changes to the C&C supertagger training process. As a state-of-the-art high performance system, the code was highly optimised and complex. I implemented two perceptron-based algorithms for model estimation and parallelised the entire training process including feature extraction and model estimation. These changes made it possible to train on vast amounts of data. Without the parallelisation, RAM usage would have been a major bottleneck. Without the perceptron-based algorithms, training would have been far too slow for large scale exploration. Models were trained using the final system that had two orders of magnitude more features, and others used three orders of magnitude more training data.

To generate more training data I used the initial system to parse all of Wikipedia, and all of the Wall Street Journal data in the North American News Corpus. After filtering, described in Chapter 3, this amounted to $26,000,000$ sentences from Wikipedia and $5,349,000$ from the Wall Street Journal, far more than the $40,000$ commonly used to construct supertagging models. This new data, generated by the parser, was used to retrain the supertagger. By training models on progressively more data I show that this training method can lead to significant improvements in speed and accuracy. Not only is performance improved on newspaper text, the traditional domain for parser construction and evaluation, but also on Wikipedia. Without any human annotated data or adjustments to the parser I was able to improve performance on the web text domain. In this way, I clearly demonstrate that tags chosen by the parser are what the supertagger should be aiming to produce. By training on the parser's output we can improve performance not only in the original domain the parser was constructed for, but in other domains. The results of this work were recently accepted for publication at the Australasian Language Technology Workshop.

While the idea that the parser's output is gold training data for its supertagger is new, the idea of supertagging itself has been in use for a decade. However, in this time there has been no careful investigation of the interaction between a parser and its supertagger. The most successful systems use a series of levels to balance speed and accuracy. The initial level is very restrictive, forcing the supertagger to provide only a small set of possible tags. At this level the parser is much faster, as it has far fewer options to consider, but also has relatively low coverage as many sentences do not receive a set of tags that can be combined to form a derivation. For these sentences the system drops down a level, rerunning the supertagger with looser restrictions, providing the parser larger tag sets and more flexibility. By repeating this process several times coverage can be kept close to 100% while most sentences are parsed early on at high speed.

Determining the right set of levels is extremely difficult as a change to one will affect the sentences seen at another. Without a clear picture of this interaction, these parameters have been chosen in an ad hoc manner up until now, with only slight local optimisations. I performed the first systematic exploration of this behaviour and propose one method of optimising speed while maintaining complete coverage. Such an algorithm is needed because not only is determining these parameters difficult, they need to be determined separately for every model.

Overall this work has made the parser run 86% faster on the Wall Street Journal and 30% faster on Wikipedia, without any loss in accuracy. Models were trained with orders of magnitude more data and features than previously, and in similar or only slightly longer periods of time. Finally, the analysis

performed here is the first systematic investigation of the behaviour of the parser and supertagger. These improvements will translate directly into improvements in systems for question answering, machine translation, anaphora resolution, and many other tasks in NLP. Without improving the speed of parsers we would be severely crippling these systems, limiting the maximum attainable performance.

CHAPTER 2

Literature Review

The work I have completed this year focuses on a particular aspect of the Natural Language Processing pipeline – supertagging, labelling words with a detailed description of their role in the sentence. This idea has developed over the past fifteen years and is an integral part of the parser I used, without which it would be considerably slower. However, before exploring the previous work in this field there are a few terms that need to be explained:

: The Penn Treebank

The 'Penn Treebank' (PTB) (Marcus *et al.*, 1993a) is a collection of documents annotated with POS tags and syntactic trees. It contains a range of documents, but when it is mentioned here I am generally referring to the Wall Street Journal sections, which are based on newspaper content from 1989.

: Hidden Markov Models

Hidden Markov Models (HMMs) are used when we can observe a series of emissions from a system and are trying to predict the states that the system moved through while making those emissions. For supertagging we are effectively observing a system that emits words, which we can see, and is moving through states corresponding to supertags, which we want to determine. The method works by considering the complete set of possible tags for each word and two sets of probabilities, transition and emission probabilities (Baum and Petrie, 1966). Transition probabilities measure the chance of a particular series of tags preceding the current one. Emission probabilities measure the chance of the current word being emitted if a particular tag is chosen.

: The Viterbi Algorithm

One algorithm for using HMMs to choose tag sets is the Viterbi algorithm (Viterbi, 1967). It moves through the sentence, determining the probabilities of all the possible tags for the current word and then using them, the transition and emission probabilities, to determine the

probability of each possible tag for the next word. Once the end of the sentence is reached the most probable tags can be assigned to the last word, and the paths that led to them can be traced back through the sentence to determine the sets of tags for the other words. A more sophisticated alternative is the Forward-Backward algorithm, so called because once reaching the end of the sentence it repeats the process in the other direction. The set of transition probabilities are changed to be based on the words that follow a given word while the emission probabilities remain the same.

2.1 Grammars

Two main classes of grammars have been used to try to understand natural language, phrasal and lexicalised grammars. Phrasal grammars generally define a small set of labels that capture the syntactic behaviour of a word in a sentence, such as noun and adverb, and then use a large set of rules to define how the words interact. These rules can then be used to construct a phrase structure tree in which the leaves are the words and the internal nodes are applications of rules. Lexicalised grammars take a different approach, providing a much larger set of labels, or *categories*, and only a few rules. The categories provide a more detailed description of a word's purpose in a sentence, leaving less work for the rules that determine how categories combined to form the parse tree.

One example of a lexicalised grammar formalism is Combinatory Categorial Grammar (CCG) (Steedman, 2000). In CCG there are two types of categories. The first type are *atomic* and are one of S, N, NP, PP, for clauses, nouns, noun phrases and prepositional phrases respectively. The second type of category is complex and contains two parts, an argument and a result, denoted by either 'Result / Argument' or 'Result \ Argument'. The slashes indicate whether the Argument is expected to lie to the right or left respectively, and the result and argument are categories themselves (atomic or complex). These categories are then combined according to seven rules, forward and backward application, forward and backward composition, backward crossed substitution, type raising and coordination, some of which are demonstrated below.

Figure 2.1 presents two examples of sentences and their CCG derivations. In both examples the line directly beneath the words contains the categories that were assigned to each word, NP for `I`, (S\NP)/NP for `ate` and so on. The lines that follow show a series of rule applications, building up the parse tree. These examples demonstrate three types of rules. The lines with a $>$ sign at the end indicate forward

application, in which a complex category combines with the category to its right. This is possible when the complex category is of the form 'Result / Argument' and its argument is the same as the tag to its right, such as in the first derivation when the (S\NP)/NP category for `ate` combines with the NP category for `pizza` to produce an S\NP category. The lines with a $<$ sign at the end are showing backward application, which is the same idea, but in the opposite direction. This can be seen in the last step of each example, when the S\NP category combines with the NP category to its left to form an S category.

Note in particular the change of tag for `with` in the two examples and its affect on the subsequent rule applications. In the first case the words `with cutlery` combine with `ate pizza`, as shown by the combination of the S\NP and (S\NP)\(S\NP) categories. This makes sense since the words `with cutlery` are adding extra information to the verb `ate`. However, in the second case `with anchovies` is combined with `pizza` to form a single noun phrase before being combined with `ate`. Again, this make sense, since `with anchovies` is describing a property of `pizza`. Now consider replacing `anchovies` with another word, such as `coffee`, in which case the correct analysis would be the first one, despite the two words being similar in the sense that they are both a form of food. People are able to distinguish between these two cases through the use of extra knowledge about the regular toppings on pizza, but in general a parser does not have such extra knowledge to draw upon. This ambiguity, called prepositional phrase attachment, is one of the reasons parsing is so difficult.

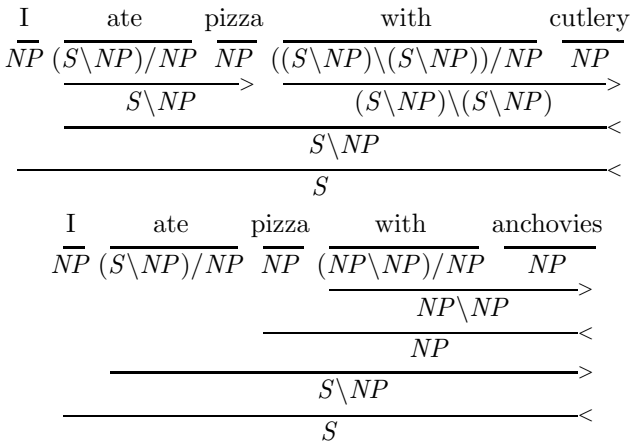


FIGURE 2.1: Example CCG derivations for two sentences.

The CCG parser used in this work is the C&C parser (Clark and Curran, 2003, 2007b). The C&C parser applies labels to words using a variant of the Viterbi algorithm for Hidden Markov Models, and then determines the correct series of rules to apply through use of the CKY chart parsing algorithm and

	Accuracy			Speed	
Parser	Precision	Recall	F-score	Time (min.)	Sentences per second
Charniak	89.5%	89.6%	89.5%	28	1.4
Collins	88.3%	88.1%	88.2%	45	0.9
Sagae	87.5%	87.6%	87.5%	11	3.6
Hockenmaier	84.3%	84.6%	84.4%	Not Given	Not Given
CCG	88.3%	87.0%	87.6%	1.9	21.0

TABLE 2.1: Performance comparison over section 23 of the Penn Treebank for a range of parsers (Clark and Curran, 2007b; Sagae and Lavie, 2005; Hockenmaier, 2003). Precision and recall are calculated over sentences receiving a parse only. The first three parsers are evaluated on accuracy of labeled constituents in the Penn Treebank, while the last two are evaluated on predicate-argument dependencies in CCG-bank, a CCG annotated corpus that uses the same base text as the Penn Treebank.

dynamic programming. The results in Table 2.1 demonstrate that it is a particularly efficient state-of-the-art parser.

As Figure 2.1 demonstrates, for a given word, the choice of tag is highly dependant on its context, and may require world knowledge (such as the regular toppings for pizzas). Without this knowledge, parsers must consider all possible parses, and since this could apply to every word in the sentence, it leads to a time complexity for parsing of at least $O(n^3)$, and in the case of CCG, $O(n^6)$. In most applications time is an important constraint, meaning that at current speeds most parsers cannot feasibly be used, regardless of how accurate they are. Consider the speeds shown in Table 2.1. The most accurate parsers are able to parse approximately one sentence per second. At this speed a typical novel containing 5,000 sentences on average 20 words long, would take over 80 hours to parse. Clearly, to parse the entire English canon using one of these parsers would take an unacceptable amount of time, let alone parsing a corpus such as Wikipedia or working online, responding to users.

For CCG the parsing process is preceded by 'Supertagging', where the initial categories are assigned to each word, such as the tag (S\ NP)/NP for the word `ate` in Figure 2.1. However, before this occurs there is another stage, Part of Speech (POS) tagging. POS tags are the familiar classes of words taught in primary school - nouns, verbs, adjectives, and so on. Even at this stage there is some degree of confusion, as many words are included in multiple classes, such as the word 'tag', which can be used as a noun (the correct tag is noun), or a verb (the program will correctly tag all verbs). Determining the correct tag for a given word requires examination of its context. Note that in Figure 2.1 the word `with` would be labelled with the same POS tag, preposition, in both cases.

Supertagging can improve the speed of parsers by decreasing the range of possible categories for each word in the sentence. As in Part of Speech tagging, this is achieved by considering a lexicon of tags each word could be assigned and reducing each set based on the surrounding context. The difference is that the sets of POS tags used in phrasal grammars are orders of magnitude smaller than the sets of supertags used in lexicalised grammars.

By reducing the number of tags to consider for each word, supertaggers leave parsers with far fewer possible derivations to consider. In the ideal case, when a supertagger is able to select a single tag for each word, the parser only needs to determine how to combine the tags to form a valid parse. The syntactic content of supertags has led to the description 'almost parsing' (Joshi and Bangalore, 1994) for supertagging because of the great reduction it causes in the range of possible parses.

The structure of supertaggers is a balance between performance and complexity. If the tagger is too simple its performance will suffer. If it uses greater information and more complex algorithms it will be slower, effectively moving the time cost of parsing from the parser to the tagger, without an overall improvement. Most supertaggers use algorithms with complexity $O(n)$ and only consider a small window around the word being tagged. Commonly used features include the surrounding words, their POS tags and, if already assigned, their supertags. This combination of extremely efficient algorithms and local context leads to a great saving of work for parsers, at very little extra cost.

The actual tagging process can be performed in a variety of ways. Initially supertaggers were used to choose a single tag, specifically the most common tag for the given word in the given context in the training data. This has the disadvantage that if the set of tags given does not lead to a valid parse, the parser is unable to consider alternatives. The alternative, providing multiple possible tags, has its own problems. The more tags that are assigned, the greater the number of possible derivations that the parser has to consider and the slower it will be. However, if the number of tags assigned is decreased too much we return to our original problem of accuracy losses. Striking a balance between speed and accuracy is difficult and ultimately it would be preferable to improve supertagging so that only a small set is required to attain high accuracy.

As with most tasks in Natural Language Processing, the training data is a crucial part of model development, and the available gold-standard annotated data is limited. One way of overcoming this challenge, which has been successfully applied to classifiers, is 'semi-supervised training'. This term covers a

range of methods that all involve using a system to automatically label more data, expanding the training dataset. However, since the amount of unlabelled data available is orders of magnitude larger than the amount of labelled data, these methods require a scalable, high performance structure to be utilised effectively.

2.2 Supertagging

Super-tags were first proposed by Joshi and Bangalore (1994) as the equivalent of POS tags for Lexicalized Tree-Adjoining Grammar (LTAG). As previously mentioned, the difference between POS tags and supertags is that the latter contain much more detailed syntactic information. To provide this extra information the sets of supertags must be much larger. Usually a supertag set contains on the order of hundreds of tags. One automatically extracted set of tags for LTAG had 3964 tags (Chen *et al.*, 2002). Most POS tag sets contain less than fifty possible tags. For instance, the Penn Treebank uses only thirty-five (Marcus *et al.*, 1993b).

When supertagging, even once the set of tags available for each word is cut down to those observed in training data or defined by a lexicon, the set of tags that could be assigned to each word is still large. The first supertaggers selected a single tag for each word based on its local context. To handle data sparseness words were not used; instead a POS tagger was run first and n-grams of surrounding POS tags were used to define the context of a word (Joshi and Bangalore, 1994). The examples in Figure 2.1 clearly demonstrate that this solution to the data sparseness problem will lead to mistakes, as this model would see the same set of POS tags for each sentence, and therefore label `with` incorrectly in one of the cases.

To experiment with a supertagger Chandrasekar and Bangalore (1997a) incorporated an n-gram supertagger into an information retrieval system. The system was designed to identify sentences related to ‘appointments’, such as ‘John Smith was appointed chairman of the board’. The supertagger was used to identify noun and verb chunks in each sentence. These were used to form ‘augmented patterns’ for relevant and irrelevant examples. The same method was applied to each sentence in the test set and if a pattern was observed that had been identified during training, the sentence was classified accordingly. This system was also used to perform a comparison between POS tags and super tags. The POS tags were used to form similar ‘augmented patterns’ and tested in the same way. As expected, the extra

information provided in the supertags led to less generalisation, reflected by fewer incorrect identifications of appointments, but also more instances of appointments being missed. Overall supertags were demonstrated to be more effective (Chandrasekar and Bangalore, 1997c). Supertagging was slower than POS tagging by approximately a factor of two, but this is still much faster than full parsing, and the supertagger always returns an answer, where as a parser may not be able to determine a parse in some cases.

This system was also used to consider how much context is most effective (Chandrasekar and Bangalore, 1997b). By varying the amount of context used between one and four words either side of the word being considered it was shown that specifying more context improves precision, since the supertagger is more certain about the context being considered, but decreases recall as data sparseness becomes a greater issue. Based on F-Score measurements Chandrasekar and Bangalore found that one or two words either side was the most effective feature set. These observations fit with the intuitive idea that a more specifically defined context will leader to greater certainty, but less generalisation.

One issue for assigning a single tag is that if it does not lead to a valid parse the parser has no other alternatives to consider. Chen *et al.* (1999) discussed the possibility of assigning a set of tags, 'multi-tagging', as well as experimenting with long distance features based on preceding phrase heads. Two methods for defining classes of tags were considered; context based classes, and confusion classes. Context based classes were defined by sets of tags that were observed in the presence of the same set of features in the training data. Confusion classes were defined by running the supertagger on a small, annotated data set and placing tags that were assigned incorrectly into the same class as the true tag.

Assigning multiple tags to each word raises the question of what order the tags should be considered. Chen *et al.* (2002) approached this question by using a trigram based supertagger to choose multiple tags, and the Viterbi algorithm to determine the most likely sequence. Then, instead of associating each word with a single tag from the most likely path, each word was associated with the n tags that had the highest prefix probabilities. First the system was evaluated without re-ranking the tag sets by passing them to a parser and counting the number of sentences successfully parsed. As expected, increasing the size of the tag sets led to an increase in the supertagger's accuracy and the number of parsed sentences. However, when more than four tags were assigned parsing was rendered infeasible due to time constraints. After establishing the benefits of more accurate supertagging, Chen et al investigated re-ranking based on a range of features, demonstrating improvements of over 1%. Clearly, the more

accurate the set of supertags provided the higher coverage will be, but if this improvement is achieved by supplying more tags it incurs a speed penalty.

The effect of supertagging on parsing efficiency demonstrates that lexical ambiguity is an important factor in parsing complexity (Sarkar *et al.*, 2000). However, this method of increasing efficiency often comes at a cost of coverage and Sarkar et al showed that the accuracy of these supertaggers on automatically extracted LTAG grammars is too low for successful integration into a full parser.

2.2.1 Supertagging for Combinatory Categorial Grammar

Supertagging was first applied to Combinatory Categorial Grammar (CCG) by Clark (2002), who performed comparisons with POS taggers and LTAG supertaggers. Rather than using a Hidden Markov Model, the supertagger determined the probability of each word being assigned each possible tag through Conditional Maximum Entropy Modeling. This method was chosen because it provides greater flexibility when adjusting the feature set and makes it easier to define a multi-tagger. Rather than defining a fixed number of tags to be produced per word the supertagger included all tags whose probabilities were within some factor, β, of the highest probability category. Similar settings were applied to an LTAG supertagger (Chen *et al.*, 1999) and the CCG supertagger was shown to perform worse for single tagging, but better for multi-tagging. As for LTAG supertaggers, the use of supertagging for CCG improved the speed of the parser, with a greater improvement for smaller tag sets, though with a slight loss in coverage.

The weights for features in the maximum entropy model for the CCG supertagger were estimated using Generalised Iterative Scaling (GIS) (Darroch and Ratcliff, 1972), a method chosen because it is a very simple algorithm that often outperforms more efficient algorithms in practise. Curran and Clark (2003) showed that a correction feature was not required to guarantee convergence, even when the sum of the feature values for each event is not constant. Additionally, it was shown that by applying a Gaussian prior instead of a frequency cutoff additional features can be incorporated without causing over-fitting. This is useful because the extra features may allow the supertagger to be more confident in the tags chosen, and possibly lead to a reduction in the number of the tags assigned to each word.

By tightly integrating a supertagger with a CCG parser, Clark and Curran (2004) were able to achieve great improvements in speed without sacrificing accuracy. The crucial development that made this possible was the use of feedback between the parser and supertagger during parsing on both training

Supertagger	Accuracy	Evaluation Corpus	Av. Tag Set Size
Joshi and Bangalore (1994)	77.3%	100 sentences from the WSJ	1
Chandrasekar and Bangalore (1997c)	89.4%	20,000 words from the WSJ	1
Bangalore (1997)	92.2%	50,000 words from the WSJ	1
Chen *et al.* (2002)	82.99%	Section 22 of the PTB	1
Chen *et al.* (2002)	90.42%	Section 22 of the PTB	2
Chen *et al.* (2002)	94.19%	Section 22 of the PTB	8
CCG, Clark (2002)	90.5%	Section 23 of CCG-bank	1
CCG, Clark and Curran (2004)	97.0%	Section 00 of CCG-bank	1.4
CCG, Clark and Curran (2004)	98.5%	Section 00 of CCG-bank	2.9
CCG, Clark and Curran (2004)	99.1%	Section 00 of CCG-bank	21.9

TABLE 2.2: Comparison of supertagger accuracy, including a range of settings for multitaggers.

and test data. To improve efficiency only local features were used, eliminating the need for the Viterbi algorithm. During training the supertagger was used to select a set of plausible but incorrect tags, to which the correct tag was added, producing a set for the parser similar to those that would be produced by the supertagger on unseen data. When parsing unseen data the supertagger initially used a large β value, and while the parser was unable to determine a parse the value was gradually decreased, expanding the tag sets. These methods increased the speed of the C&C parser by a factor of seventy-six, making it an order of magnitude faster than comparable systems. As Table 2.2 shows, the C&C (2004) supertagger produced much more accurate tag sets for a range of tag set sizes.

Supertagging has subsequently been combined with a full LTAG parser (Sarkar, 2007). The influence of supertagging on parsing efficiency was again demonstrated, and the use of co-training the supertagger on the parser's output was shown to be more effective than entirely supervised training methods.

2.3 Summary

Supertaggers have been used effectively in a range of NLP tasks, such as information retrieval and parsing. By reducing the set of possible lexical categories for each word in a sentence they provide a crucial source of information that can dramatically improve efficiency. While they are not perfect and in some cases will provide incorrect tags, through multi-tagging and tight integration a balance of speed and accuracy can be achieved that is beneficial overall.

However, accurate supertagging is only possible when multiple tags are supplied, and every extra tag that is assigned as a possibility increases the number of derivations the parser must consider. Any decrease

in ambiguity should translate directly to an increase in speed, but simply supplying fewer tags using current models would lead to a decrease in accuracy. Instead we should aim to reduce the number of tags provided that the parser does not use, ie only supplying one tag for each word – the tag that the parser would have used anyway.

CHAPTER 3

Evaluation

Before describing the experiments performed it is important to establish the baseline system and the methods that will be used to evaluate all changes. This chapter also describes the development of the extra training data that is used throughout.

3.1 Data

This work has used two forms of data, newspaper text and web text. Specifically, I have used 5.4 million sentences from the Wall Street Journal (WSJ) between 1987 and 1996 in the North American News Corpus (NANC), and 26 million sentences from a 2009 dump of the English section of Wikipedia. The first set was chosen as the standard parser evaluations are performed on newspaper text from the WSJ. Wikipedia was chosen because web text, and Wikipedia in particular, have recently become very popular resources in NLP. Also, while Wikipedia and newspaper text are definitely different, Wikipedia is still predominantly made up of well formed sentences, as opposed to a great deal of other web text, making it a sensible first step towards parsing web text.

3.1.1 Training

The standard corpus for parser training and evaluation is the Penn Treebank (PTB) (Marcus *et al.*, 1993b), a collection of documents annotated with POS tags and syntactic trees. I have used CCGBank (Hockenmaier and Steedman, 2001), a translation of the bracketing structure of the Penn Treebank to the CCG formalism (Steedman, 2000). For training I have used section 02-21, which consist of $39,604$ sentences originally from the 1989 WSJ. No such corpus was available for Wikipedia.

To produce more training data I automatically labelled the resources described above. The raw text was tokenised using the NLTK tokeniser (Bird *et al.*, 2009), and parsed using the C&C parser and models

version 1.02[1]. To ensure no overlap occurred between our automatically labelled WSJ data and the CCGBank data all of the 1989 data in the NANC was excluded. For the Wikipedia data I applied a set of simple rules to exclude instances as described below:

- At least 6 tokens
  List entries - "Discography."
- At most 125 tokens
  Entire lists that appear as a single sentence
- At least one word must start with a lowercase letter
  Titles - "The Davis Chinese Christian Church."
- The first character must be a letter
  Strange data artifacts - ";CD Two."
- The first token cannot start with ‘Category’
  Structure information - "Category:WikiProject New Hampshire articles."
- The last three tokens cannot form "refer to ."
  Disambiguation page headings - "... may refer to ." and "... can refer to ."

The automatically labelled data from both corpora was also altered slightly due to parser limitations:

- Removed non-printable characters
  For example, "... for a cost of \xc2\xa31.50." became "... for a cost of 1.50."
- Translated from utf-8 to ASCII
- Replaced all vertical bars with colons as vertical bars are used as separators in the parser input

And finally, all sentences longer than 250 tokens were excluded from the NANC data. The sentences this affected were generally long lists of scores or stock market results.

3.1.2 Testing

For the WSJ I used section 00 of CCGBank to evaluate accuracy. The annotations in CCGBank include gold standard grammatical relations for all sentences, as well as the lexical category for each token. These 1,913 sentences allowed us to measure the overall change in parser performance, as well as supertagging performance. For Wikipedia I used a recently developed set of 300 sentences annotated

[1] http://svn.ask.it.usyd.edu.au/trac/candc

with grammatical relations and 1000 sentences labelled with lexical categories.(Clark *et al.*, 2009) The WSJ and Wikipedia–300 sets contain 45422 and 6696 word–category pairs respectively. It is also worth noting that these sets were created by running the supertagger, then manually correcting the answers, which may have introduced some bias in favour of the supertagger.

To measure speed accurately I needed much larger sets. I set aside $10,000$ sentences from the 1988 WSJ in the NANC for these measurements, excluding them from the training data described above. Similarly, $10,000$ sentences of Wikipedia data were excluded from the training set and used for speed evaluation.

3.2 Metrics

Before considering the metrics used it is important to understand the concept of 'beta levels'. When choosing the tags to assign the C&C supertagger determines a ranking for the possible tags for each word. Assigning only the top tag does not lead to high enough accuracy to enable wide coverage parsing. Rather than assigning a fixed number of tags, the top three for example, the C&C supertagger assigns a variable number based on a probability distribution expressing confidence. Specifically, every tag that is within a certain fraction, beta, of the most likely tag is included. This distribution varies for every word and its context, so the number of categories assigned to each word by the supertagger depends heavily on the beta level used.

This presents a problem because the more categories that are assigned, the higher the tagging accuracy, and the appropriate beta levels for a model depend highly on the properties of its weight distribution. To fairly compare the models presented here I have used the test set to determine values for the five beta levels that will lead each model to produce the same number of tags per word on average. The values are tuned on section 00 of the WSJ, using the C&C parser and models 1.0 to determine reference ambiguities. These beta levels are then used for all tests, including Wikipedia.

For the models trained on Wikipedia data the change in ambiguity between the WSJ and Wikipedia is considerably smaller than for the reference model. I chose not to re-tune on a sample of Wikipedia sentences as this would mean I am effectively using two versions of each model during evaluation.

For accuracy three standard metrics are used, precision, recall and F-score. These are defined in terms of:

- True Positives, instances that occur in the system output and the true output

- False Positives, instances that occur in the system output, but not the true output
- True Negatives, instances that did not occur in the system output or the true output
- False Negatives, instances that did not occur in the system output, but did occur in the true output

$$\text{Precision} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalsePositive}} \tag{3.1}$$

$$\text{Recall} = \frac{\text{TruePositive}}{\text{TruePositive} + \text{FalseNegative}} \tag{3.2}$$

$$\text{F} - \text{score} = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3.3}$$

The F-scores given are calculated based on comparisons with gold standard labelled dependencies. The category accuracies are for the first beta level only, and a word is considered correctly tagged if any of the assigned categories is correct.

Parsing speeds were calculated using the parser's internal timers, measuring the overall parse time. As the amount of training data scales up, so too does the time it takes to train models. One of the benefits of the methods described in the following chapter is that they can train considerably faster, to demonstrate these benefits I measured the amount of time spent training various models. All speed measurements were performed using a 3GHz Intel Core 2 Duo CPU, and 4Gb of RAM.

3.2.1 Significance Testing

Statistical significance testing was performed to determine if changes in model performance were meaningful or not. The test applied reports whether two systems' responses are drawn from the same distribution, where scores of 0.05 and lower are considered significant (Chinchor, 1992).

The test works by taking two sets of output on the test set and randomly interchanging the entries ten thousand times, counting whether the difference in a given metric between the two sets has increased. If the two sets of responses are from different distributions, randomly mixing them should bring all of the

Data	Word Accuracy (%)	Sentence Accuracy (%)
WSJ Section 00	96.49	51.23
WSJ Section 02-21	97.96	67.61
WSJ Section 23	96.90	55.01
Wikipedia 300	98.7	79.3
Wikipedia 1000	97.86	75.50

TABLE 3.1: Accuracy of the C&C POS tagger.

metrics closer together, as the two sets of results move to some intermediate distribution. If the two sets are from the same distribution, then the change could be in either direction due to random variations. If the change is observed to increase less than 5% of the time the distributions are considered different[2].

3.3 Baseline Performance

The baseline system is the C&C parser, revision 1300[3]. The supertagging model used was trained using section 02-21 of CCGBank. It was estimated using the BFGS algorithm (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970) over 500 iterations.

Before considering the performance of the baseline supertagger and parser it is worth considering the C&C POS tagger. The POS tags assigned to the sentence form more than half of the features used by the supertagger, and so their accuracy is crucial. In Table 3.1 we can see that the tag accuracy is quite high, but nevertheless, a large proportion of sentences contain at least one error.

The default settings for the parser include five beta levels and corresponding tag dictionary cutoffs, as shown in Table 3.2. The first set of ambiguity measurements shown here are the values that the rest of our models are tuned to. Note the decrease in accuracy when tagging Wikipedia, despite an increase in the number of categories assigned per word (and hence increased chances of supplying the correct category). This is expected, as the supertagger was trained on newspaper text, not Wikipedia. But it is an important fact, since it demonstrates the decrease in performance when switching domains.

From Table 3.3 we can make several important observations. First, the influence of POS tagging accuracy is made absolutely clear by the drops in F-score of 2.4 and 1.7 percent, despite much smaller drops

[2]The actual script used is based on David Vadas' Python implementation of http://www.cis.upenn.edu/ dbikel/software.html#comparator

[3]Revision number refers to the svn repository at http://svn.ask.it.usyd.edu.au/candc/trunk

		WSJ			Wikipedia		
		Ambiguity	Accuracy (%)		Ambiguity	Accuracy (%)	
Beta	Dict Cutoff	(Cats/Word)	Word	Sentence	(Cats/Word)	Word	Sentence
0.075	20	1.270	96.07	59.54	1.314	95.37	49.33
0.03	20	1.429	96.76	64.45	1.511	95.82	54.00
0.01	20	1.718	97.36	69.21	1.853	96.58	57.67
0.005	20	1.983	97.59	70.88	2.178	96.85	59.67
0.001	150	3.576	98.44	78.78	4.111	98.0	70.0

TABLE 3.2: Baseline supertagging accuracy using POS tags produced by the C&C POS tagger.

in supertagging accuracy – 1.5 and 0.9 percent respectively. We can also observe the importance of supertagging, as both tests in which gold standard supertags are provided have f-scores over 90%. Clearly improvements in the POS tagger and supertagger can translate into significant gains in parser accuracy.

However, the most important observation for this work relates to the speed measurements. In the first two rows of each section the parser and supertagger are interacting, with the supertagger supplying limited tag sets and then gradually adding more tags if the parser cannot find a spanning analysis. Even at the first level the number of tags assigned is around 1.3, leaving the parser with many combinations of tags to consider. When the number of tags per word is reduced to 1, in the bottom two rows of each section, the system is considerably faster. This is because the parser has less work to do in the first place, since there is only one combination of tags to work with, and since there are no other ambiguity levels to consider it never has to try parsing a sentence multiple times before finding a derivation or giving up.

This clearly indicates that if we can reduce the number of tags per word that the supertagger supplies we will obtain an increase in speed. The third row of each section represents an oracle supertagger that can perfectly assign tags. The fourth row represents a supertagger that can second-guess the parser, producing the set of tags it would have used anyway. One means of working towards either of these systems is to use more data when generating our supertagging model. For the oracle supertagger we would need more gold standard data, which is expensive and time consuming to generate. Meanwhile for the supertagger that second-guesses the parser we need more data labelled with the parser's output, something we can easily generate by simply running the parser on large amounts of text. If we can use this data to construct better models of what the parser wants we could improve parsing speed by a factor of between three and five.

Data	Supertag Accuracy (%) Single	Multiple	Ambiguity (cats / word)	F-score (%)	Speed (sent / sec) Eval Data	10k
WSJ Section 00						
Gold POS tags	92.59	97.34	1.27	85.79	68	*
Auto POS tags	91.14	96.07	1.27	83.41	68	48.54
Gold Supertags	n/a	n/a	1	96.81	250	*
Parser Supertags	92.07	n/a	1	83.41	230	290.2
Wiki Sent 300						
Gold POS tags	90.7	96.4	1.3	84.2	58	*
Auto POS tags	89.8	95.4	1.3	82.5	58	46.31
Gold Supertags	n/a	n/a	1	91.0	300	*
Parser Supertags	92.0	n/a	1	82.6	280	253.2

TABLE 3.3: Baseline model performance.

3.4 Summary

The most important aspect of this chapter is the illustration of potential speed improvements. The baseline system has an F-score of 83.41 on the WSJ, and 82.5 on Wikipedia, and can process 48.54 sentences of the WSJ per second, and 46.31 sentences of Wikipedia per second. If the supertagger could cut down the tag sets it currently supplies to the one tag per word that the parser actually uses in the final derivation, the system would process sentences more than five times faster. This clearly demonstrates that successfully training the supertagger to provide what the parser wants will provide significant efficiency improvements.

CHAPTER 4

Algorithms

A range of methods are used by supertaggers to choose the sets of tags to assign to each word. The two main areas of variation are the set of features used, and the model that determines the implications of those features. To enable the use of greater numbers of more sophisticated features supertaggers need to become scalable. More features also mean more parameters to be optimised in our models, so our optimisation algorithms need to be efficient. Tackling the challenge of algorithms that are more efficient, but just as accurate, is the focus of this chapter.

4.1 Background

The C&C supertagger selects tags using a Hidden Markov Model and the Viterbi algorithm, which are described at the start of Chapter 2. One issue not dealt with in that section was how the transition and emission probabilities are determined. These values are somehow defined by the language and domain being considered, but are unknown to us. The methods described below provide a means of estimating them.

4.1.1 Maximum Entropy Modeling

The general problem of how to use several different probability estimates to form one that captures them all has been extensively studied by statisticians. The maximum entropy model proposed by Jaynes (1957) works by reformulating the estimates as constraints on the expectation, or average, of various functions. Then, the probability distribution that satisfies all these constraints and has the highest entropy is selected as the new model.

One of the first uses of maximum entropy modeling in NLP was by Lau *et al.* (1993) for language modeling. The parameters of the model were estimated using the Generalised Iterative Scaling method (GIS),

and the result was better than other state of the art methods at the time. Perhaps the most well known use of these models in NLP is by Ratnaparkhi (1996) in a state of the art POS tagger. The core idea of this work was to make better use of contextual information, made possible by the flexibility of maximum entropy modelling. The POS tagger produced had an accuracy of 96.5% on a subset of the Wall Street Journal section of the Penn Treebank, was flexible and only required a lexicon of possible POS tags for each word[1]. Subsequently maximum entropy models have been incorporated into a range of systems, such as the supertaggers described previously (Curran and Clark, 2003).

However, McCallum and Pereira (2001) highlighted what they called the *label bias problem* for maximum entropy Markov models. The concept behind this problem is that decisions made in one state, such as when assigning a POS tag to a word, are not influenced by the choices to follow. It has also been shown that variants of iterative scaling perform quite poorly compared to general function optimisation algorithms such as the conjugate gradient method (Malouf, 2002). Despite these issues, maximum entropy modeling has proved very effective for estimating weights for supertagging models.

4.1.2 Perceptrons

Perceptrons are another means of determining values that can be used as probabilities in HMMs. They were first proposed as a model for human learning (Rosenblatt, 1958), based on investigation of the structure of the brain and identification of its similarity to the structure of computers. Since then perceptrons have been extensively studied as a simple machine learning algorithm for classification of linearly separable data.

Recently, a variant of perceptrons was proposed by Freund and Schapire (1999) which stores all prediction vectors considered during training, along with a count of the steps they were maintained for. This count is then used as a weight for the vector, on the assumption that better vectors will last longer since they misclassify fewer instances. These weighted vectors are then combined in a weighted majority vote to create the final prediction vector. The result is a simple, easy to implement algorithm, which was shown to be competitive with Support Vector Machines[2] on a handwritten digit classification task.

Collins (2002) showed that these methods could be applied to tasks in NLP, such as NP Chunking and POS tagging, with better performance than maximum entropy models. Specifically, using a voted perceptron and trigram features for training, a Viterbi based system had an F-score of 93.53% for NP Chunking and

[1] Without such a lexicon accuracy was reduced by 0.12%

[2] An influential machine learning algorithm that been shown to be very effective.

$$\min_{\bar{\tau}} \frac{1}{2} \sum_r || \bar{M}_r + \tau_r \bar{x}^t ||_2^2$$
$$\text{subject to: (1) } \tau_r \leq \delta_{r,y^t} \text{ for } r = 1, ..., k$$
$$\text{(2) } \sum_{r=1}^{k} \tau_r = 0$$

FIGURE 4.1: Equations for the MIRA update scheme.

an error rate of 2.93% for POS tagging, compared to 92.65% and 3.28% respectively for a similar system trained with a maximum entropy model. Interestingly, the perceptron model also achieved its best result after far fewer iterations, between six and forty, as opposed to the one hundred to two hundred required for the maximum entropy model.

This perceptron method was applied to parsing by Collins and Roark (2004), using an incremental beam search parser, which works by developing a set of linear constraints, one for each incorrect parse in the training data. The parser performed similarly to another based on a generative model, with F-scores of 87.8% for data with gold standard POS tags, and 86.6% when tags were generated by a tagger. Similar methods were recently applied to the C&C parser (Clark and Curran, 2007a), leading to performance comparable to a log-linear model, but with much lower system requirements. Importantly, since the perceptron based model is on-line, the final model differs slightly depending on the order of the training data, but Clark and Curran (2007a) showed that this did not influence the performance of the parser.

4.1.3 Margin Infused Relaxed Algorithm

The Margin Infused Relaxed Algorithm (MIRA) also follows the standard multi-class perceptron algorithm, but applies a different update method. The intention is to make the smallest possible change to the weights such that the correct class would be produced by a specified margin. As defined by Crammer and Singer (2003), the update function adjusts the weights by a set of values satisfying:

Where τ is the update to be made, $\bar{M}$ is the matrix of weights, $\bar{x}^t$ is the value of the feature, k is the number of classes, and δ is the Dirac delta function, equal to 1 only when r is the index of the correct classification.

4.1.4 The C&C Parser and Supertagger

The C&C parser uses the CKY algorithm to construct the 'chart', an efficient representation of all possible analyses for a sentence. The most probable derivation is found using the Viterbi algorithm and probabilities are calculated based on a conditional log-linear model.

The supertagger uses a maximum entropy based model to assign a set of possible lexical categories to each word in the sentence. The baseline system estimated the model using either GIS or BFGS and ran on only one processor.

If the supertagger assigns only one category to each words its accuracy is too low to be effectively incorporated into a parser. By multitagging we can make the supertagger more accurate, but at the cost of speed as the parser must consider larger sets of possible categories. The beta levels define cutoffs for multitagging based on the probabilities from the maximum entropy model. If the parser is unable to form a spanning analysis the beta level is decreased and the supertagger is rerun. The exact values of these levels greatly influences parsing accuracy and speed. Accuracy is decreased in two ways: by not providing enough categories at any level, leading to no spanning analysis; or by providing too many, causing an 'explosion' in the chart.[3]

The initial feature set used for tagging included unigrams of POS tags and words and bigrams of POS tags, all in a five word window surrounding the word being tagged. The weights for these features were estimated by either Generalised Iterative Scaling (GIS) (Darroch and Ratcliff, 1972) or the Broyden-Fletcher-Goldfarb–Shanno method (BFGS) (Broyden, 1970; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970).

4.2 Implementation

4.2.1 Averaged Perceptron

The standard multi-class perceptron maintains a matrix of weights, containing a row for each attribute and a column for each class. The weight in cell (c, r) indicates how strongly related the attribute r and the class c are. When all attributes are binary valued the class is assigned by ignoring all attributes that do not occur and determining which column has the greatest sum. During training the class that column corresponds to is compared to the true class and if it is correct no change is made. If the predicted class is incorrect the weights are updated by subtracting 1.0 from all weights for the predicted class and adding 1.0 to all weights for the true class. The averaged perceptron (AP) follows the same algorithm, but returns the average of the weight matrix over the course of training, rather than its final state.

[3] An 'explosion' is when the chart exceeds a particular size, which for this work was set to $300,000$ total categories (ie the initial supertags and all the intermediate categories that are part of the derivation).

The reason the average is more effective than the final value is that the perceptron only converges for linearly separable data, which our data is not. This means that the perceptron will jump between states, optimising for the most recent set of observations, potentially at the cost of accuracy on a large set of previous observations. Therefore the final state of the perceptron is highly dependent on the most recently observed training instances. By taking the average of all states we will be creating a state that is most similar to the states that are most often correct.

4.2.2 Margin Infused Relaxed Algorithm

We have applied a slight variation of the MIRA update method that can be expressed as follows [4]:

$$\min(\max, \frac{\text{margin}+\sum_f p_w - t_w}{|\text{features}|(1+\frac{1}{\text{n}_{\text{above}}})})$$

Where $margin$ is the absolute difference that will be created between the true classification and those that previously ranked above it, the sum is over all features, p_w and t_w are the weights associated with the feature f for the predicted and true classes respectively, $|features|$ is the number of active features, and n_{above} is the number of categories that had higher sums than the correct category. The constant, max, was introduced to prevent a single event causing extremely large changes to the model.

We also made it possible to enable shuffling between iterations of the algorithm. The idea for this was to prevent the model from overfitting to the particular order of training instances.

4.3 Results

Using larger datasets for training can take a prohibitive amount of time for the GIS and BFGS algorithms. However, any time benefits provided by other algorithms need to be balanced with their influence on accuracy. Table 4.1 shows how the accuracy of the algorithms described previously compare with the previous estimation methods.

Tagging and parsing accuracy are both similar, and the small decrease that does occur in F-score is not statistically significant. The speed of the parser using these models is also similar, with the averaged perceptron outperforming the previous methods by a considerable margin.

[4] The derivation of this alternative form can be found here:
http://www-inst.eecs.berkeley.edu/ cs188/sp09/projects/classification/classification.html

Algorithm	Cat Accuracy (%) Single	Multiple	F-score (%)	Speed (sent / sec)
GIS	91.49	96.32	83.82	51.7
BFGS	91.38	96.29	83.73	52.1
AP	91.41	95.65	83.74	59.2
MIRA	91.42	96.19	83.69	50.6

TABLE 4.1: Performance comparison of model estimation algorithm on the WSJ.

Data (sentences)	Training Time (sec) GIS	BFGS	AP	MIRA
40,000	7,200	6,300	76	96
80,000	14,000	13,000	160	200
440,000	*	*	950	1,200

TABLE 4.2: Comparison of training time for several model estimation algorithms.

The time saving provided by the perceptron based algorithms is clearly illustrated by Table 4.2. The new algorithms are able to train on eleven times as much data as the previous methods, in a sixth of the time. The final row is included to demonstrate that the time taken continues to scale linearly with respect to the amount of training data. Comparable GIS and BFGS models were trained on $440,000$ sentences, but a different computer was used, making direct comparison inappropriate.

Further comparisons of the algorithms can be found in Chapter 5. They are not included here as the experiments rely upon the adaptive training methods still to be described.

4.4 Summary

By implementing two perceptron based algorithms to estimate the parameters for our maximum entropy models I have been able to utilise orders of magnitude more data, at no extra cost in time. Importantly, even when using only the data used previously, the models produced by this training process can be just as accurate as models estimated by GIS or BFGS. These developments provide the architecture needed for rapid adaptive training on large amounts of training data.

CHAPTER 5

Adaptation

Once the extra data had been created and the new estimation algorithms were implemented it was possible to begin scaling up the amount of training data. I describe this training process as 'adaptive training' as the extra data was annotated by the baseline system, and that data is now being used to retrain an earlier stage in the system, the supertagger. To support the extra training data I also needed to parallelise the training process, and some investigation was performed to determine the scalability of the new system.

In this chapter two domains are explored, newspaper text and web text. The first is the standard domain for parser evaluation and is the domain the baseline system was trained on, and the second has recently become a particularly popular resource for many NLP tasks. Also, by considering two domains and measuring cross-corpus performance I was able to demonstrate that domain adaptation is occurring. This chapter also contains further comparison of the various model estimation algorithms to demonstrate that the averaged perceptron and MIRA continue to produce comparable results as the amount of data scales up.

5.1 Background

5.1.1 Semi-supervised Training

Most statistical parsers and supertaggers share a similar training method in which a large collection of sentences with gold-standard annotations are used to construct a model. This method has the disadvantage that it is expensive to apply to novel domains and other natural languages. An alternative is a form of semi-supervised training in which multiple models are trained on a smaller set and then reliable output from one model on unlabeled data is used as further training data for the other models.

One of the first demonstrations of semi-supervised training in NLP was for word sense disambiguation (Yarowsky, 1995), in which a model was proposed that considered collocations in documents to classify

examples of words into sense categories. Initially a small set of collocations for the word being considered were identified and labeled with a sense, and then all instances of those collocations in the corpus were labeled with the same sense. The system then labeled as much of the corpus as possible based on the context of words already labeled. By repeating this process all instances in the corpus were labeled, with an accuracy of more than 96%, clearly demonstrating the feasibility of semi-supervised training.

This was followed by Blum and Mitchell (1998), who applied co-training to classification of web pages. Two naive Bayes classifiers were used, one trained on the words in hyperlinks, the other trained on the words in web pages. Initially both were trained using a set of labeled examples. Then a series of iterations were performed in which both classifiers considered a set of unlabeled examples and added the ones they could most confidently label to the training set. For evaluation the classifiers were compared to a baseline classifier that always returns 'negative'[1] and another pair of naive Bayes classifiers trained on only the initial training set. Both of the co-trained classifiers performed better, in particular the page based classifier, which more than halved its error rate. Interestingly in one series of iterations the page based classifier became worse than the baseline during the first ten iterations, but was then able to improve during the next thirty iterations, more than halving its initial error rate. This clearly demonstrates that once enough iterations have been completed, co-training provides a great improvement, successfully utilising unlabeled examples. Blum and Mitchell (1998) also developed a formalism to describe the learning process in terms of a bipartite graph. Two sets of vertices are defined by different views of the documents; in the test described one set was the words in hyperlinks and the other was the words in pages. Examples from the data set form edges in the graph, indicating that the two features that are linked should indicate the same label. Once enough edges have been added, the graph will consist of two components, corresponding to the two labels.

Algorithms for supervised learning have been developed, such as AdaBoost (Freund and Schapire, 1997), which combine the results of several weak learning algorithms to produce improved results. Collins and Singer (1999) proposed a co-training variant of AdaBoost, CoBoost, and applied it to a document classification task. Unlike previous work, in which all of the models would change in each iteration, Collins and Singer (1999) alternated between models, using one to train the other and vice versa. Starting with a small set of seven seed rules, the two classifiers gradually developed by adding rules based on new classifications of previously unlabeled documents. By only adding the most definite new rules in each iteration the system was able to outperform a Yarowsky (1995) style implementation by 10%.

[1] 78% of the pages had a correct classification of negative.

One disadvantage of the approaches to co-training described so far is that they assumed that two redundant views of the data existed that could be used individually for perfect classification. Goldman and Zhou (2000) considered using models that did not guarantee this property, and instead required that the models divided the examples into a set of equivalence classes. In each iteration the system determined which of the recently labeled examples to pass from one model to the other by hypothesis testing on confidence intervals, the amount of extra labeled data that would be obtained, and a conservative estimate of the classification noise rate. On the sample task described in the paper, categorical classification of UCI datasets, co-training using the ID3 and HOODG algorithms led to an error rate more than 4% lower than either algorithm alone. This is an improvement of more than 25% over either algorithm alone, and 15.6% over an algorithm that omnisciently picks the better result from the two algorithms. This last result clearly indicates that addition of extra training data through co-training has led to more accurate models. Importantly, the original source of this training data was unlabeled data, which is available in large quantities.

5.1.2 Semi-supervised Training for Parsers

Semi-supervised training was first considered for parsing by Sarkar (2001), who applied co-training to a supertagger and parser for LTAG. Unlike previous attempts to apply unsupervised training to parsing, the evaluation was performed using sentences in the Penn Treebank[2]. In each iteration the sentences that were parsed with greatest certainty were added to the training set. A major difference between parsing and the previous co-training experiments described is that the set of 'classes' that the parser can place sentences in is arbitrarily large, and while the set used by the supertagger is limited by a lexicon, it is still orders of magnitude greater than the sets previously described. Despite these challenges, the co-trained supertagger and parser improved by more than 7% in both precision and recall over the baseline parser trained by supervised methods.

Another form of semi-supervised training is self-training, in which a single system is trained on its own output on unlabeled data. Usually this has been found to provide either only slightly positive or significantly negative results, presumably because errors in the original model are amplified in subsequent models (Charniak, 1997). However, McClosky *et al.* (2006) demonstrated that a variant of self-training similar to co-training can provide improvements. Rather than using two completely independent systems that are retrained in each iteration, a parser and reranker were used and only the parser was retrained. The purpose of the reranker was to take the top fifty parses produced by the parser and rerank them

[2]Previously successful attempts had generally used shorter, less complex sentences.

according to a range of linguistic features. Note that this does differ from self-training approaches, since the parser's output is modified independently before being reused, but it is not quite the same as co-training since only one of the models is being changed in each step. Using these methods the reranking parser was able to improve its F-score from 90.3% to 92.1% on section 22 of the Wall Street Journal corpus. These results are important as the core idea of their approach is the same as the idea being applied here, except that instead of using the output of a parser and reranker to retrain the parser, I am using the output of a supertagger and parser to retrain the supertagger.

5.1.3 Semi-supervised Training for the CCG Supertagger

There has been considerable investigation of the features used by the models in supertaggers, but how they are trained has not yet been as thoroughly addressed. Both of these areas are critical to supertagger performance, which has been extensively demonstrated to have a great influence on subsequent processing. One method that has been successfully applied elsewhere to improve training when only limited labelled data is available is semi-supervised training, but it has not yet been applied to a CCG supertagger. One of the benefits and challenges of semi-supervised training is the large quantity of extra training data it provides. This is a benefit because it makes training on domains with small amounts of annotated data feasible, but is also a challenge because it creates a demand for scalable high performance systems.

Another idea that has not been addressed is to consider using semi-supervised training of a supertagger to adapt it to a particular parser. If we aim only to improve speed, while maintaining accuracy, then one method of doing so is to decrease the number of tags supplied, while still including the one that the parser was going to use anyway. This should lead to the same accuracy, as the parser uses the same tag it would have if the model assigned more tags. However, the parser is faster as it has fewer derivations to consider.

Of course it would be ideal to improve the confidence of our model using vast amounts of gold standard annotated data, reducing ambiguity while still including the correct tag. This would give a speed improvement, as there are fewer derivations for the parser to consider, and probably an accuracy improvement too, as the tagging is more accurate. However, while we do not have access to vast amounts of gold standard annotated data, we can easily generate vast amounts of data labelled with the parsers final derivations. This data can then be used to retrain the supertagger to give the speed boost described in the previous paragraph.

To enable the use of large amounts of data for semi-supervised training the supertagger needs to be scalable. To achieve this aim I needed to parallelise the training process.

5.1.4 Message Passing Interface

The Message Passing Interface (MPI) is a message passing application programmer interface that was created to assist programmers in developing code that will be portable while maintaining high performance (Snir *et al.*, 1995). The actual definition of MPI is language independent, and since its creation it has been implemented across a range of platforms and languages including C (Gropp *et al.*, 1996) and Python (Miller, 2002).

Despite being used extensively elsewhere, very little work appears in the literature for NLP that uses MPI. One of the first examples is by Clark and Curran (2003), where it was used to parallelise the GIS parameter estimation algorithm to allow a larger dataset to be held in memory (30 GB of RAM). This work was also included in the outline of a high performance infrastructure for NLP (Curran, 2003), which described using MPI to provide scalability as corpora grow in size.

Recently other groups have started to use MPI, such as Kazama and Torisawa (2008), who parallelised a clustering algorithm for constructing a gazetteer for named entity recognition. This made it feasible to perform clustering with a large vocabulary and a computationally expensive algorithm. As well as MPI, there has been some use of other platforms for parallel computing in NLP, such as the use of Grid computing in Hughes *et al.* (2004) for creating indices for information retrieval tasks.

The low usage of MPI in the field could be the result of the limited size of annotated corpora available for training. However, as was shown earlier, recent developments in semi-supervised training using a large amount of unannotated data are creating a need for efficient, parallelised implementations of tools in NLP.

5.2 Implementation

To enable the use of larger models I increased the amount of accessible RAM and processing power by parallelising the supertagger training using MPI and the MapReduce library MRMPI. The stages in the training process for the system can be seen in Figure 5.1. Parallelising this process involved

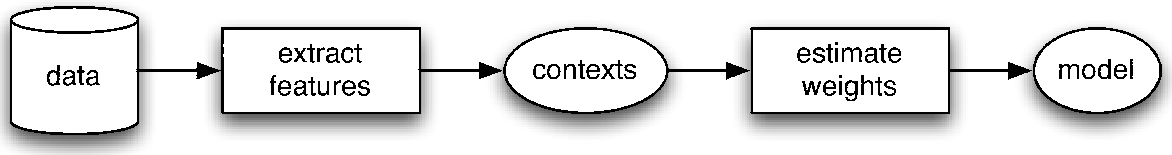


FIGURE 5.1: Single thread model creation.

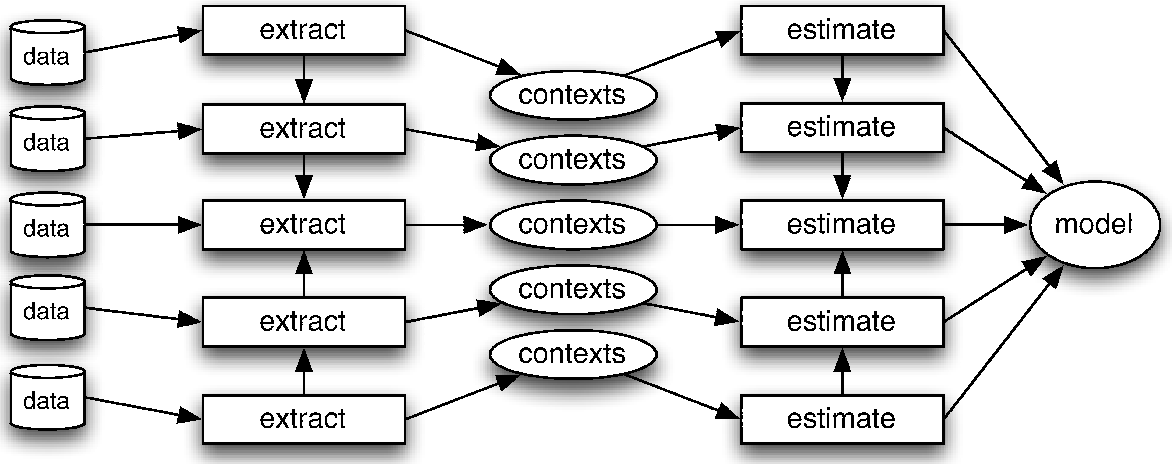


FIGURE 5.2: Parallel model creation.

modifying the entire process, which can be divided into two primary stages, feature extraction and weight estimation.

5.2.1 Parallelising Feature Extraction

The first stage of supertagging is feature extraction and aggregation. Extraction is trivial to parallelise by dividing the contexts amongst a set of computers. Aggregation is necessary to determine which features should be excluded as being too rare and so I used MRMPI to combine frequency counts and update the values for relevant nodes[3]. This process is summarised in the first half of Figure 5.2, where the data is divided initially, and then there is communication between nodes during feature extraction so that the sets of contexts and features created are correct.

[3]The parallelised form of feature extraction was implemented by pair programming with Jessika Rosener during the JHU CLSP Workshop 2009

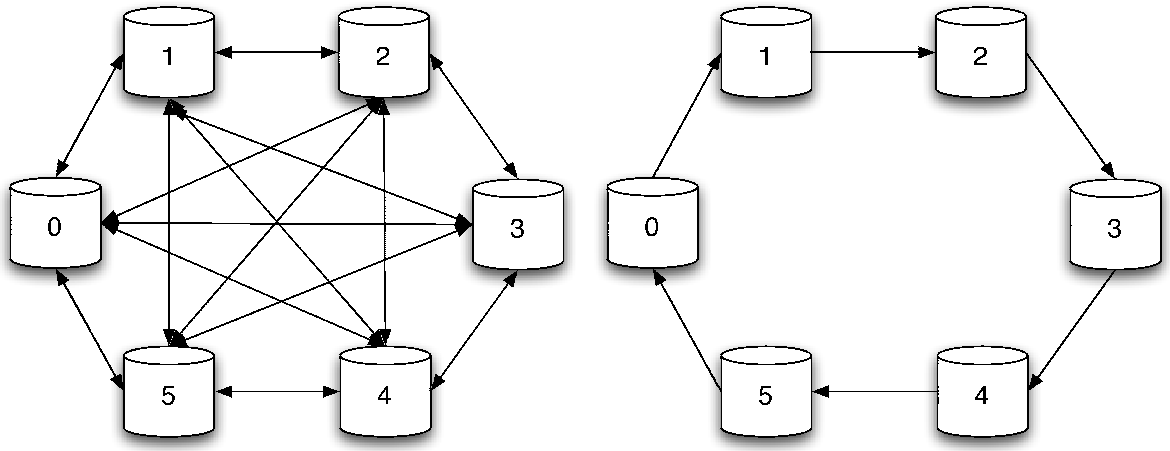


FIGURE 5.3: Information flow for parallel estimation of maximum entropy models and perceptron models

5.2.2 Parallelising Feature Weight Estimation

For weight estimation the maximum entropy methods were 'embarrassingly parallel', as the main processing is the calculation of sums of weights across all training instances. The parallel version of these methods differ in three main ways. First, the instances are divided between a set of computers. Second, sums are calculated across all computers to determine necessary changes to weights. And third, after each update the changes are distributed to all nodes.

The perceptron methods adjust the weights based on each training instance individually and so the parallelisation above was not applicable. The training instances are still distributed across a cluster of computers, but during weight estimation only one computer is working at a time, adjusting the weights based on all of its instances before passing the updated weights to the next node. This saves time by removing the cost of loading the training instances from hard disk when there are too many to fit in RAM.

The difference in communication usage by the two methods is captured by Figure 5.3. Clearly the maximum entropy methods involve more network usage, but unlike the perceptron methods, every node is involved in the calculations, leading to considerable time savings as the number of nodes scales up.

	Time (1000s of seconds)							
	1 process per node				All on one node			
Data	1	2	3	4	1	2	3	4
40k	2.9	1.8	1.4	1.2	2.9	1.8	1.4	1.2
80k	6.9	4.0	3.0	2.5	6.9	4.0	3.0	2.5
440k	38	21	15	12	38	21	15	12
2000k	170	94	66	52	170	95	67	53
4000k	*	190	130	110	*	*	*	*

TABLE 5.1: Small scale scalability tests of the parallel GIS implementation.

5.3 Results

5.3.1 Scalability

Since the purpose of the parallel implementation was to improve the scalability of the training process a series of tests were performed with different numbers of processors and amounts of training data to confirm that the changes were effective. All of the results in this section were collected by training models using newspaper text and GIS, with the exception of the MIRA estimated models in Table 5.3. The system that was used for training the models was the Silica computing cluster, which is made up of seventy-four computer nodes. Each node contains eight cores, four each from two Intel Xeon X5355s, clocked at 2.66GHz, as well as sixteen gigabytes of RAM.

While it was important to perform the tests summarised in this section, it would have been a waste of resources to perform more tests than necessary. As a result, not all of the cells in the tables contain results. Any cell containing a '–' was not tested for this reason. Cells containing a '*' indicate tests that could not be run because insufficient RAM was available.

The first set of tests were small scale, to see at what point a single node has insufficient RAM to support all of the data, and how whether inter-node network latency was an issue. Table 5.1 clearly demonstrates that the parallel implementation is providing a significant improvement. The left half of the table describes tests where one MPI node was running on each compute node, though a request was made for all eight cores to ensure full access to the RAM and no interference by other users' processes. The right hand side describes tests where all of the MPI nodes were on a single compute node. There is little to no difference between the two sides, which is good, as it means the distribution of MPI instances across the cluster can be of any form, as long as enough RAM is available.

	Time (1000s of seconds)						
Data	1	2	4	8	16	32	64
40k	2.9	1.8	1.2	0.98	0.83	0.81	0.78
80k	6.9	4.0	2.5	2.0	1.4	1.2	1.2
440k	38	21	12	8.0	6.1	4.6	3.6
2000k	170	94	52	–	–	–	–
4000k	*	190	110	–	–	–	–

TABLE 5.2: Large scale scalability tests of the parallel GIS implementation.

	Time (1000s of seconds)							
	GIS				MIRA			
Data	1	2	4	8	1	2	4	8
40k	2.9	1.8	1.2	0.98	0.12	0.12	0.11	0.12
80k	6.9	4.0	2.5	2.0	0.27	0.26	0.25	0.26
440k	38	21	12	8.0	2.0	1.8	1.8	1.8
2000k	170	94	52	–	9.7	8.8	8.8	–
4000k	*	190	110	–	*	20	–	–

TABLE 5.3: Comparison of training time for parallel implementations of GIS and MIRA.

Also, the results show significant improvements in training time. The relationship between number of instances and amount of time is not quite directly inversely proportional, but doubling the number of instances does lead to at least a 43% decrease in time for the larger data sets.

Once I had established that inter-node communication would not be a major issue I performed the larger scale tests summarised in Table 5.2. As the number of instances scales up, we experience diminishing returns, but as the amount of data scales up those returns improve. With sixty-four instances we can train on eleven times as much data as a single instance and take less than 25% longer.

As described in the previous section, the parallel implementations differ as the perceptron algorithms modify the weight vector continuously, and so only a single processor can work at once. However, the parallel implementation is still important, as it allows access to far greater amounts of RAM. In Table 5.3 we can see that increasing the number of instances does not improve training time for MIRA, but the algorithm is so fast anyway, that it far outpaces GIS at these scales.

Based on these tests I came up with different strategies for running large scale training experiments:

Data	Supertag Accuracy (%) Single	Multiple	F-score (%)	Speed (sent / sec)
WSJ	91.49	96.32	83.82	51.7
NANC				
40k	90.81	95.57	83.02	**59.1**
400k	91.57	96.09	83.55	57.2
2000k	91.71	96.35	83.85	56.4
4000k	91.70	96.39	83.97	55.7
5349k	91.73	96.39	84.03	55.7
WSJ + NANC				
40k	91.62	96.36	84.05	56.9
400k	91.87	96.46	83.98	56.5
2000k	91.98	96.64	84.22	55.6
4000k	91.98	**96.68**	**84.32**	55.0
5349k	**92.02**	96.67	**84.32**	55.1

TABLE 5.4: Performance of models trained using NANC data.

- GIS and BFGS, as many instances as possible, without using up RAM (extra usage due to process overheads[4])
- AP and MIRA, one instance per node, using as few nodes as possible

5.3.2 North American News Corpus

Before considering adaptation to other domains it was important to investigate the potential for improvement on newspaper text. Table 5.4 presents the results of adaptive training experiments on the North American News Corpus. All of these models were estimated using GIS and are evaluated here on Section 00 of the WSJ.

The most important result here is the clear improvement in speed from 51.7 to over 55 sentences per second. These improvements are far smaller than those demonstrated in Chapter 3. Presumably to attain the speeds from Table 3.3 a reduction in ambiguity will be required. The relationship between ambiguity, accuracy and speed is explored further in the following chapter.

Table 5.4 also demonstrates that adaptive training does not decrease accuracy. In fact, slight improvements in tagging accuracy and parsing F-score are observed. The best result is an improvement of 0.5%,

[4]These overheads are in fact the reason that no tests were performed with 8 instances on a single node for two million sentences.

Data	Supertag Accuracy (%) Single	Multiple	Ambiguity (cats / word)	F-score (%)	Speed (sent / sec)
WSJ	90.05	95.34	1.32	82.5	46.8
Wikipedia					
40k	90.56	94.79	1.26	82.1	**61.3**
400k	90.89	95.71	1.27	82.7	**61.3**
2000k	91.13	95.80	1.28	82.7	57.3
4000k	91.07	95.82	1.28	83.0	59.9
8000k	91.16	95.80	1.28	**83.3**	60.5
WSJ + Wikipedia					
40k	90.64	95.37	1.29	82.6	58.9
400k	91.02	95.68	1.28	82.5	59.7
2000k	**91.23**	95.73	1.28	82.4	59.7
4000k	91.11	**95.83**	1.28	82.7	59.1
8000k	91.16	95.82	1.28	82.9	59.8

TABLE 5.5: Performance of models trained using Wikipedia data.

which was made by the model trained on almost all of the WSJ data in the NANC and is statistically significant.

As expected, the addition of gold standard training data leads to a considerable improvement in accuracy. Interestingly, even when tagging accuracy is similar, such as for the model trained on $2,000,000$ sentences from the NANC and the model trained on $40,000$ sentences plus section 02-21 of the WSJ, the model trained with gold standard data performs better.

5.3.3 Wikipedia

One of the primary aims of this work was to use supertagger adaptation to improving parsing performance on domains other than newspaper text. To improve performance on web text I trained new supertagging models using automatically labelled Wikipedia text, the results of which can be seen in Table 5.5. The improvements here are even greater than for the previous tests, with increases in parsing speed from 46.8 to around 60 sentences a second, with no loss in accuracy.

Interestingly, as more data is used parsing speed does not improve, but accuracy does considerably. The best performing model, trained using eight million wikipedia sentences, improves by 0.8%. Importantly, this change was found to be statistically significant using the test described in Chapter 3.

	Supertag Accuracy (%)							
	WSJ				Wikipedia			
	Single		Multiple		Single		Multiple	
Training Corpus	no gold	with WSJ	no gold	with WSJ	no gold	with WSJ	no gold	with WSJ
WSJ	n/a	91.49	n/a	96.32	n/a	90.05	n/a	95.34
40k NANC	90.81	91.62	95.57	96.36	90.20	90.53	95.10	95.28
400k NANC	91.57	91.87	96.09	96.46	90.47	90.58	95.74	95.65
2000k NANC	91.71	91.98	96.35	96.64	90.73	90.74	96.13	96.01
4000k NANC	91.70	91.98	96.39	**96.68**	91.04	91.04	96.16	96.19
5349k NANC	91.73	92.02	96.39	96.67	90.93	91.04	**96.25**	96.21
40k Wiki	88.75	91.55	93.90	96.31	90.56	90.64	94.79	95.37
400k Wiki	89.89	91.34	95.07	96.22	90.89	91.02	95.71	95.68
2000k Wiki	90.42	91.34	95.54	96.27	91.13	91.23	95.80	95.73
4000k Wiki	90.52	91.33	95.64	96.29	91.07	91.11	95.82	95.83
8000k Wiki	90.66	91.36	95.75	96.29	91.16	91.16	95.80	95.82

TABLE 5.6: The effect of adaptive training on supertagging accuracy.

In the previous section the same dataset was used to tune the beta levels as was used for the evaluation – Section 00 of the WSJ. Here I also tuned on Section 00, but tested on the Wikipedia–300 dataset. Interestingly, this causes a significant difference in ambiguity, leading the supertagger to assign fewer tags per word. This makes sense, since these models were trained on the Wikipedia domain and therefore should be more confident when tagging Wikipedia text. However, decreasing ambiguity leads to a decrease in supertagging accuracy, and lower supertagging accuracy generally leads to lower parsing accuracy. This makes the improvements in tagging accuracy and parsing F-score even more impressive.

5.3.4 Cross-Corpus Evaluation

To determine whether the supertagger is actually adapting to the new domain or simply improving overall, in this section I present cross-corpus comparison of the models in the previous two sections. Each table considers a different aspect of performance, and is structured with the cross-corpus results in the top right and bottom left quadrants. To demonstrate the influence of the addition of gold standard data each row actually contains the results for two models, one with only the data in the 'Training Corpus' column, and another with the addition of the gold standard WSJ data.

5.3.4.1 Supertagging Accuracy

The first performance measure to consider is also the most difficult to interpret – supertagging accuracy, shown in Table 5.6. If all of our data was gold standard the results of this section would be straightforward to interpret, as the more effectively a model is learning from the data, the higher its accuracy should be. For automatically labelled data the situation is more complicated as we are not in fact developing models to produce the correct answer, but rather to produce the answer the parser would choose given all the options it was previously.

For the WSJ section of the table the results are entirely as expected. The Wikipedia trained models consistently perform worse, the addition of gold standard WSJ data always boosts performance, and training on more of the NANC data leads to improved performance. Importantly, adding almost any amount of Wikipedia data decreases performance in comparison to training on the WSJ alone, which fits with the theory that our model is adapting to the Wikipedia domain.

The results for Wikipedia are where the effect of adaptive training become more difficult to interpret. The ambiguity of each beta level was tuned to be the same for all models for the WSJ, but this does not translate to the same ambiguity for Wikipedia. The WSJ model and all the NANC models had tagging ambiguities of between 1.32 and 1.33 for Wikipedia, while the Wikipedia trained models had ambiguities between 1.26 and 1.29, as shown in Table 5.5. This is the most probable explanation for the slightly poorer performance of the Wikipedia trained models when multitagging. However, the single tagging results, which sidestep the ambiguity issue, clearly indicate that the Wikipedia trained models are performing better. Again, this demonstrates that the supertagger is successfully adapting to the new domain. Additionally, the NANC trained models perform worse on Wikipedia than they do on the WSJ, which fits with the adaptation hypothesis.

5.3.4.2 Parsing Accuracy

The next metric to consider is parsing accuracy, shown in Table 5.7. While the primary aim of this work is to improve parsing speed, it would not be a worthwhile improvement if it came at the cost of parsing accuracy. Additionally, it was expected that training on extra data from one domain should improve performance on that domain in particular, and not another.

The results in Table 5.7 clearly demonstrate that performance has not decreased and that the improvements are domain specific. For the WSJ the most accurate models are those trained on the NANC, while

	F-score (%)			
	WSJ		Wikipedia	
Training Corpus	no gold	with WSJ	no gold	with WSJ
WSJ	n/a	83.82	n/a	82.5
40k NANC	83.02	84.05	81.5	81.8
400k NANC	83.55	83.98	81.2	81.4
2000k NANC	83.85	84.22	81.6	81.6
4000k NANC	83.97	**84.32**	81.9	81.9
5349k NANC	84.03	**84.32**	82.0	81.8
40k Wiki	79.83	83.90	82.1	82.6
400k Wiki	81.75	83.69	82.7	82.5
2000k Wiki	82.57	83.70	82.7	82.4
4000k Wiki	82.82	83.77	83.0	82.7
8000k Wiki	82.97	83.75	**83.3**	82.9

TABLE 5.7: The effect of adaptive training on parsing accuracy.

almost all of the Wikipedia trained models perform worse than the baseline. This suggests that the Wikipedia trained models have adapted and no longer model the newspaper text domain as effectively. For Wikipedia we observe the opposite, with almost all Wikipedia trained models performing better than the baseline, and all NANC trained models performing worse.

The results for the Wikipedia trained models are particularly impressive, as in Table 5.6 we saw that the decrease in ambiguity was leading to lower tagging performance. However, these seemingly contradictory results make sense when we consider the fact that the models are being trained on parser output. We would expect that adaptive training of the models would make them produce the tag sets that the parser chooses. This could lead to a decrease in tagging performance, since the model is not learning to produce the gold standard. However, it should not lead to a decrease in parsing performance, since the tagger is now even more likely to provide the tag that the parser would have used anyway.

Interestingly, performance continues to improve as more training data is used, right up to the largest models, though we do observe diminishing returns.

5.3.4.3 Parsing Speed

The final metric, but most important for this work, is speed. Table 5.8 clearly shows that the models are adapting to the domain they are trained on. In fact, some of the models perform even worse than the baseline in the cross-corpus tests.

	Speed (sent / sec)			
	WSJ		Wikipedia	
Training Corpus	no gold	with WSJ	no gold	with WSJ
WSJ	n/a	51.7	n/a	46.8
40k NANC	**59.1**	56.9	48.5	45.9
400k NANC	57.2	56.5	47.7	47.5
2000k NANC	56.4	55.6	49.2	49.7
4000k NANC	55.7	55.0	48.6	48.6
5349k NANC	55.7	55.1	49.7	48.9
40k Wiki	48.1	54.3	**61.3**	58.9
400k Wiki	46.9	50.6	**61.3**	59.7
2000k Wiki	45.0	47.9	57.3	59.7
4000k Wiki	44.9	45.6	59.9	59.1
8000k Wiki	46.1	47.5	60.5	59.8

TABLE 5.8: The effect of adaptive training on parsing speed.

It may seem surprising that speed improvements do not occur as progressively more data is used, but this is probably a result of the tuning of beta levels. By tuning to match the average number of tags supplied we are effectively tuning the amount of work the parser will have to do as it works out which combination of tags leads to the best derivation. This issue will be further explored in the following chapter.

Of course, this argument raises the question of why any improvement is observed. The reason appears to be that more sentences are parsed earlier, as shown in Table 5.9. Parsing a sentence at an earlier level is a serious advantage, as going to the next level would mean repeating the entire parsing process, and doing so with more possible derivations, as at the lower levels the supertagger is supplying more tags.

5.3.5 Algorithm Scalability

In the previous chapter we saw that the perceptron based algorithms could train on orders of magnitude more data, but given the same amount of data as the previous methods, they produced slightly worse results. Here I investigate their performance over a wider range of datasets.

The first section of Table 5.10 is exactly the same as in the previous chapter, and the other sections display similar trends. The perceptron algorithms perform worse, but not by a great amount. Importantly, once training on more than just the WSJ, MIRA and AP both perform better than the baseline for parsing accuracy in all cases.

	Number of sentences parsed at each level																			
	Without WSJ 02-21										With WSJ 02-21									
	WSJ					Wikipedia					WSJ					Wikipedia				
Data	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
WSJ	1787	41	34	13	19	278	8	8	3	3	1787	41	34	13	19	278	8	8	3	3
NANC																				
40k	1829	25	14	7	21	287	5	1	3	3	1832	23	16	7	20	279	8	4	3	4
400k	1857	17	13	3	3	286	8	2	0	0	1852	19	12	5	6	287	7	3	0	0
2000k	1867	10	9	3	3	290	5	2	0	0	1869	9	8	2	4	289	6	2	0	0
4000k	1871	11	5	4	4	293	5	1	0	1	1874	8	5	4	4	291	6	2	1	0
5349k	1872	12	5	3	3	292	4	1	1	1	1874	9	5	4	3	290	5	1	2	0
Wikipedia																				
40k	1827	29	20	8	16	286	6	4	2	2	1831	33	19	4	10	282	6	7	0	5
400k	1855	15	14	1	11	292	3	3	0	1	1853	24	12	4	4	291	4	1	2	1
2000k	1870	11	9	4	1	292	3	4	0	0	1871	14	8	2	1	292	5	2	0	0
4000k	1875	9	7	1	4	293	4	2	0	1	1872	14	6	1	3	293	4	1	0	1
8000k	1875	10	8	3	3	292	5	2	0	1	1878	10	5	3	3	292	5	1	0	1

TABLE 5.9: Number of sentences parsed at each level for a range of models.

One issue faced by the perceptron based methods is that they do not actually produce probability distributions. This is solved by normalising, but this was not possible for AP trained models that used more than $400,000$ sentences as the values being used became too large.[5]

The same trends can be seen for Wikipedia in Table 5.11. The perceptron algorithms consistently perform slightly worse, but in this case the accuracy of even the largest model is still worse than the baseline. However, as was observed in Table 5.7, even GIS does not show noticeable improvements in accuracy when trained on more Wikipedia data. Also, all of these models were trained using combinations of Wikipedia and the gold standard WSJ data, while the mos accurate GIS models were those trained using Wikipedia alone. This decision was made mainly to make these results more comparable to those in the previous table.

[5]The normalisation process involves considering exponentials of the weights, which quickly overflow as the values increase.

Algorithm	Supertag Accuracy (%) Single	Multiple	F-score (%)	Speed (sent / sec)
		WSJ		
GIS	91.49	96.32	83.82	51.7
BFGS	91.38	96.29	83.73	52.1
AP	91.41	95.65	83.74	59.2
MIRA	91.42	96.19	83.69	50.6
		WSJ + 40k NANC		
GIS	91.62	96.36	84.05	56.9
BFGS	91.51	96.24	84.14	57.3
AP	91.42	95.77	84.04	64.0
MIRA	91.61	96.25	83.93	56.6
		WSJ + 400k NANC		
GIS	91.87	96.46	83.98	56.5
BFGS	91.46	96.12	83.89	60.4
AP	91.68	96.08	83.91	62.9
MIRA	91.78	96.39	83.92	57.5
		WSJ + 2,000k NANC		
GIS	91.98	96.64	84.22	55.6
BFGS	91.95	96.52	84.31	57.8
MIRA	91.83	96.60	84.11	56.9
		WSJ + 4,000k NANC		
GIS	91.98	96.68	84.32	55.0
BFGS	91.97	96.55	84.19	57.2
MIRA	91.93	96.62	84.10	55.0
		WSJ + 5,348,997 NANC		
GIS	92.02	96.67	84.32	55.1
MIRA	91.93	96.58	84.15	54.7

TABLE 5.10: Comparison of WSJ performance for various model estimation algorithms.

Algorithm	Supertag Accuracy (%) Single	Multiple	Ambiguity (cats / word)	F-score (%)	Speed (sent / sec)
		WSJ			
GIS	90.05	95.34	1.32	82.5	46.8
BFGS	89.89	95.33	1.31	82.6	48.5
AP	89.28	94.49	1.35	81.7	57.1
MIRA	89.53	95.19	1.33	81.5	47.9
		WSJ + 40k Wikipedia			
GIS	90.64	95.37	1.29	82.6	58.9
BFGS	90.52	95.24	1.29	81.3	60.7
AP	90.32	94.61	1.28	81.7	69.7
MIRA	90.28	95.28	1.30	81.5	58.6
		WSJ + 400k Wikipedia			
GIS	91.02	95.68	1.28	82.5	59.7
BFGS	90.67	95.22	1.29	81.3	58.6
AP	90.71	95.16	1.27	82.2	69.4
MIRA	90.89	95.58	1.28	82.2	61.4
		WSJ + 2,000k Wikipedia			
GIS	91.23	95.73	1.28	82.4	59.7
BFGS	90.79	95.73	1.29	82.2	60.1
MIRA	90.76	95.62	1.28	81.9	59.3
		WSJ + 4,000k Wikipedia			
GIS	91.11	95.83	1.28	82.7	59.1
BFGS	91.01	95.85	1.28	82.2	64.3
MIRA	90.93	95.73	1.29	82.4	58.7
		WSJ + 8,000k Wikipedia			
GIS	91.16	95.82	1.28	82.9	59.8
MIRA	91.16	95.77	1.29	82.0	45.6
		WSJ + 16,000k Wikipedia			
MIRA	91.08	95.92	1.29	81.7	56.5
		WSJ + 26,000k Wikipedia			
MIRA	91.05	95.95	1.29	81.6	57.2

TABLE 5.11: Comparison of Wikipedia performance for various model estimation algorithms.

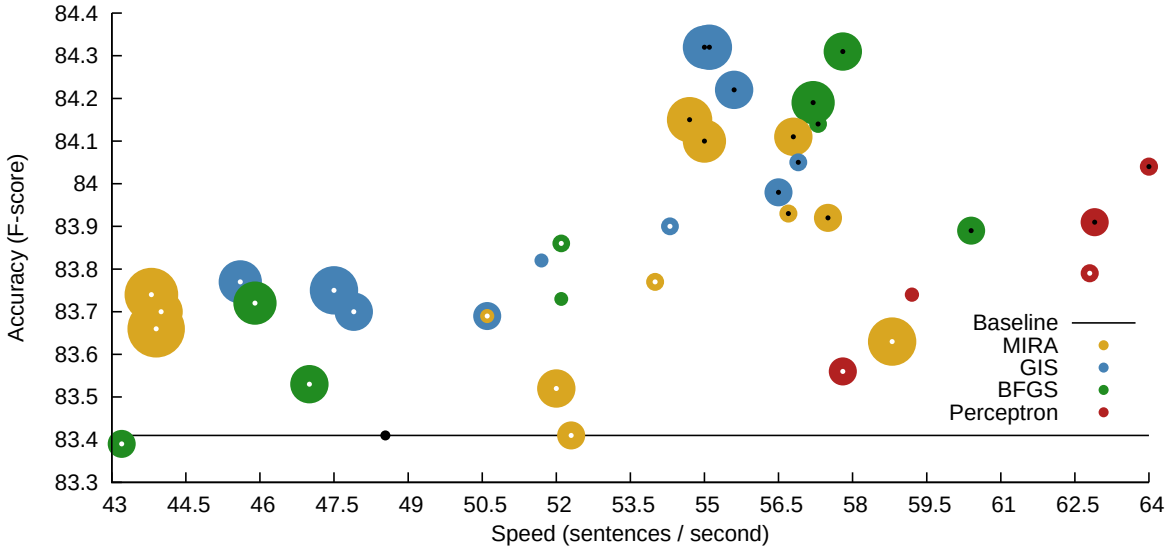


FIGURE 5.4: Overall performance comparison on the WSJ.

5.3.6 Summary

The major results of this chapter are summarised in Figure 5.4 and Figure 5.5. Before exploring the results in these figures it is important to understand all of the information they contain.

Each circle represents a different model, and its position is determined by its speed and accuracy on the corpus given in the caption. Note that the ranges covered by the axes on the two plots are different, this was necessary to improve readability.

The size of each circle reflects the amount of data used to construct the model. The sizes do not scale linearly with the amount of data, as that would make some invisible or others enormous, but they do reflect the overall trend.

The colour of each circle is determined by the algorithm used to train it, as shown in the key. Finally, the dot in the centre of each circle indicates what data was used. Black dots indicate NANC data, white dots indicate Wikipedia data, and no dot is included for models trained on only the WSJ from CCGBank.

Note that in Figure 5.4 the MIRA model trained on only the WSJ happens to coincide almost exactly with a GIS model trained on Wikipedia, at 50.5 sentences per second. The white dot at that point is for the GIS model.

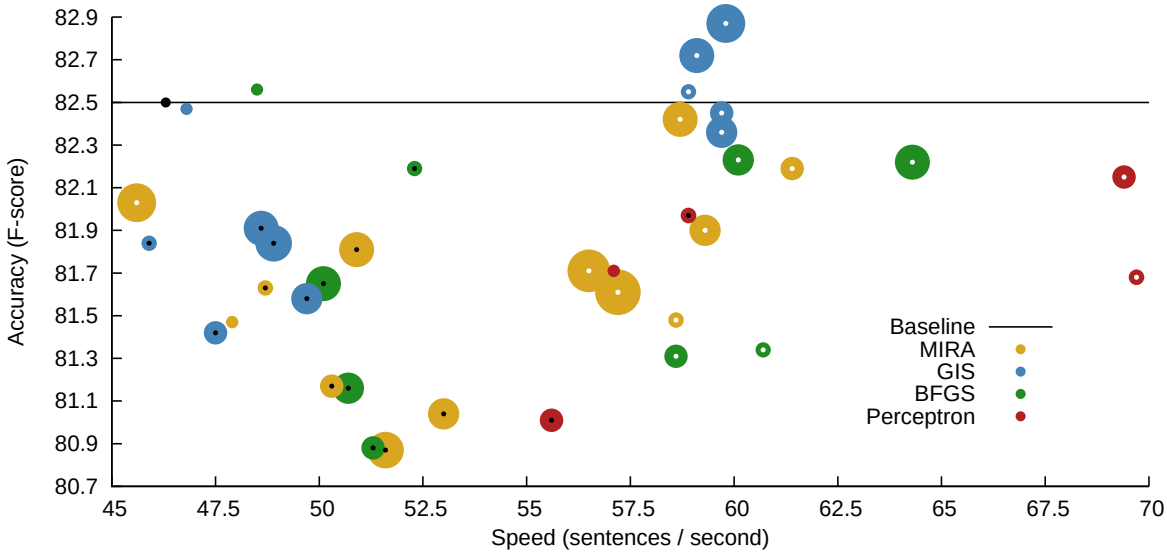


FIGURE 5.5: Overall performance comparison on Wikipedia.

The baseline model is marked using a single black dot and a black line is included at its F-score to enable easier comparison of other models.

Finally, when looking at these diagrams the best results are in the top right hand corner, where speed and accuracy are both high. However, any result that is above or close to the baseline accuracy line meets our target of maintaining accuracy.

There are a few trends to note in these diagrams. First consider the arrangement of colours. In general they are mixed up throughout the plot, which reflects the fact that all of the algorithms produce models of similar quality. This is a crucial result as it means the work described in Chapter 4 was successful.

The second important trend to consider is the distribution of Wikipedia based models compared to NANC based models. In Figure 5.4, where the evaluation is on newspaper text, the NANC based models generally perform better, clustering in the top right hand corner. The opposite trend is observed in Figure 5.4. This indicates that the models have indeed adapted to the particular domain of the training data.

Also note that in both cases the most accurate models are those trained on larger amounts of text, visible as larger circles higher up. At the same time, as more data from the wrong domain is used, performance decreases. This is further evidence that adaptation is occurring.

Some other patterns are present for the accuracy and speed of particular estimation methods. Specifically GIS appears to generate particularly good models, and the averaged perceptron appears to generate the fastest models.

5.4 Summary

I have now demonstrated that adaptive training can improve parsing efficiency. By parallelising the training process and using the algorithms described in the previous chapter I was able to train on up to 650 times as much data. These new models were produced without any extra human annotated data, and without any domain specific changes to the feature sets or alterations to the training process. The models I produced allowed the supertagger to supply the tags the parser was most likely to use, leading more sentences to be parsed earlier. This clearly demonstrates the potential for using automatically annotated data for training.

Also, by training on substantial amounts of automatically annotated data from Wikipedia I was able to create models that are adapted to the domain. Cross-corpus comparisons demonstrated that adaptation was indeed occurring, and as more training data was used the system became more accurate.

One of the major issues raised within this chapter is that by tuning on the number of tags assigned per word I am essentially tuning to the number of derivations the parser will have to consider. If the models can be tuned in a different way it may be possible to decrease ambiguity, thereby increasing speed, without losing accuracy. This challenge is one of the focuses of the next chapter.

CHAPTER 6

Optimisation and Analysis

In the process of running the adaptive training experiments two questions in particular were raised. First, with so much extra data are their more features that are common enough to be useful? And second, does a well founded method exist for optimising the number of tags assigned when multitagging? The answers to these two questions form the basis of this chapter.

6.1 Background

6.1.1 Features

Recently, Cooper (2007) experimented with a range of extra features for the CCG supertagger, with limited success. It is likely that data sparseness was a major issue for these extra features, as their greater complexity meant that each possible combination of attributes in the feature would occur less, if at all, in the limited training data. If this problem can be overcome we may be able to reincorporate these features with improved results.

6.1.2 Parser – Supertagger Interaction

When the C&C parser was first constructed it was thought that the parser should do most of the work to maximise performance. However, as described in Chapter 2, it was found that by tightly integrating the supertagger with the rest of the parser great speed improvements were possible, without loss of accuracy. This interaction makes the behaviour of the system difficult to predict, and while some attempts at local optimisation have been performed, there has been no comprehensive study of the parser – supertagger interaction.

6.2 Implementation

6.2.1 New Features

The standard features used by the supertagger, listed below, are taken from a five word window surrounding the word being tagged.

- Word unigrams
- POS tag unigrams
- POS tag bigrams

I considered the expansion of this set to include:

- Word bigrams
- Word trigrams
- POS tag trigrams

I also considered extension of the window to seven words for all features. The extra features this introduces are described as ‘far’ in the results below.

One of the issues in the current architecture of the parser is that introducing new features involves a reasonable number of modifications in various locations in the supertagging code. Until now this has not been a major issue, as more sophisticated features were not useful. Resolving this architectural issue is one potential area for future work.

6.2.2 Accurate Sentence Level Speed Measurements

To accurately measure the behaviour of the parser I introduced a new timing mechanism, using the Intel timing registers. This meant that the time taken to parse each individual sentence could be accurately measured in clock cycles. To minimise interference all timing tests were performed without a user logged in, but even so, the effects of background system processes can be observed in some of the plots as slightly darker thin vertical patches.

Features	Supertag Accuracy (%) Single	Multiple	F-score (%)	Speed (sents / sec)	Features (millions)	Attributes
			WSJ + 2000k NANC			
All	92.22	96.75	84.14	54.7	10.9	2.15
- far tags	92.21	96.76	84.16	56.0	10.8	2.15
- bitags	92.13	96.76	84.27	55.4	9.82	2.11
- far bitags	92.21	96.74	84.17	55.7	10.4	2.15
- tritags	92.11	96.75	84.26	55.8	8.51	1.91
- far tritags	92.16	96.75	84.29	55.6	9.02	2.03
Baseline	91.83	96.60	84.11	56.8	7.82	1.89
			WSJ + 4000k NANC			
All	92.28	96.79	84.25	53.4	15.5	3.15
- far tritags	92.25	96.78	84.19	54.7	13.1	3.01
Baseline	91.93	96.62	84.10	55.9	11.9	2.87

TABLE 6.1: Subtractive analysis of all-tag feature sets using four million sentences.

6.3 Feature Extension

Using the MIRA training method I was able to quickly construct a large set of models with a range of feature sets, as shown in Table 6.2.

The results are not overwhelmingly positive, but are promising. The decrease in tag accuracy when removing the extra tag based features indicates that these are the features contributing most to the improvements in accuracy. Based on these results I performed experiments with two million sentences and only the extra tag features, as shown in Table 6.1.

For these tests I switched to the NANC data as the test set for the WSJ is eight times larger than the test set for Wikipedia, enabling more rigorous evaluation. Significance testing between the baseline and the best model, all extra features except tritags, showed a statistically significant improvement in recall, but not precision or F-score. Based on these results I trained a small selection of models on four million sentences, as shown in the table. Sadly there were no further improvements in performance.

Features	Supertag Accuracy (%) Single	Multiple	Ambiguity (cats / word)	F-score (%)	Speed (sents / sec)	Features (millions)	Attributes
			WSJ				
All	90.08	95.12	1.31	82.5	42.8	8.27	5.97
- far tags	90.14	95.15	1.32	82.3	42.9	8.25	5.97
- bitags	90.05	95.24	1.32	82.5	42.1	8.07	5.96
- far bitags	89.90	95.24	1.31	82.4	43.2	8.15	5.97
- tritags	90.22	**95.34**	1.33	82.3	42.6	7.87	5.89
- far tritags	90.31	95.31	1.32	82.2	43.2	7.99	5.93
- far words	**90.32**	95.27	1.32	**82.6**	43.1	7.91	5.89
- biwords	90.16	95.19	1.31	82.4	45.4	5.09	3.59
- far biwords	90.13	95.19	1.32	82.5	43.7	7.25	5.35
- triwords	90.20	95.16	1.31	**82.6**	46.0	4.88	2.78
- far triwords	90.19	95.24	1.32	82.2	43.6	6.88	4.73
Baseline	89.53	95.19	1.33	81.5	**47.9**	0.82	0.23
			WSJ + 40k Wiki				
All	90.83	95.45	1.29	82.1	55.9	15.7	11.8
- far tags	90.68	95.45	1.29	81.9	53.6	15.7	11.8
- bitags	90.05	95.24	1.32	**82.5**	42.3	8.07	5.96
- far bitags	90.93	95.33	1.29	81.7	55.8	15.6	11.8
- tritags	90.70	95.47	1.29	82.0	54.5	15.2	11.7
- far tritags	90.95	**95.49**	1.29	82.2	54.9	15.3	11.8
- far words	90.79	95.47	1.29	81.9	55.7	15.1	11.7
- biwords	90.98	95.31	1.28	82.3	57.4	9.52	7.11
- far biwords	90.96	95.42	1.29	82.2	55.0	13.8	10.6
- triwords	**91.04**	95.42	1.29	82.1	57.8	9.02	5.48
- far triwords	90.88	95.47	1.28	82.0	55.5	13.0	9.37
Baseline	90.28	95.28	1.30	81.5	**58.6**	1.45	0.46
			WSJ + 400k Wiki				
All	91.19	95.82	1.28	82.3	57.4	65.8	51.0
- far tags	91.35	95.79	1.28	82.0	57.3	65.8	51.0
- bitags	**91.43**	95.85	1.27	82.6	56.9	65.4	51.0
- far bitags	91.29	95.79	1.27	82.2	57.8	65.6	51.0
- tritags	91.32	**95.91**	1.28	82.2	57.9	64.7	50.8
- far tritags	91.38	**95.91**	1.28	82.5	56.8	65.0	50.9
- far words	91.34	**95.91**	1.28	82.3	57.4	63.7	50.4
- biwords	91.23	95.74	1.28	**82.7**	58.3	39.8	31.9
- far biwords	91.20	95.82	1.28	82.3	57.8	57.5	46.0
- triwords	90.98	95.74	1.27	82.6	58.8	34.2	21.4
- far triwords	91.10	95.83	1.28	82.6	58.4	52.7	39.5
Baseline	90.89	95.58	1.28	82.2	**61.4**	4.62	1.61

TABLE 6.2: Subtractive analysis of various feature sets using up to four hundred thousand sentences.

6.4 The Influence of Beta Levels

The beta levels chosen have an enormous impact on the quality of the supertagger, and hence, the parser. A higher level will cut out more categories, leading to lower category accuracy and more failures to find a derivation, but faster parsing. A lower level will raise category accuracy, but slow down the parser and cause more chart explosions. For the C&C parser the situation is further complicated by the fact that multiple beta levels are chosen. This means that it may be acceptable to have a higher first level that many sentences fail to be parsed at, since those sentences will be parsed at the next level, and overall, time will be saved.

Previously the selection of beta levels has been ad hoc, with slight variations explored, but no rigorous method for their choice. In order to remain comparable to previous work all of the previous results were measured using beta levels set based on ambiguity levels. The ambiguity level is the average number of tags assigned per word by the supertagger at a given beta level. These values were calculated for the standard model and used as reference values for all others. Since the models vary greatly, the beta levels that correspond to the reference ambiguity levels had to be recalculated for each.

The problem with this scheme is clearly illustrated by Table 6.3, where a collection of models trained using various amounts of NANC data have been tested using the default beta levels, as well as with beta levels tuned as described above.

First consider the ambiguity and multi-tag accuracy columns. Most of the models have higher accuracy when using tuned beta levels, but this is not surprising, since almost all of the models have lower ambiguity at the default level, and the more tags that are supplied, the higher than chance the correct one will be included. Also, these models are being trained to produce the tag that the parser will use, rather than the true tag, and so some decrease in tag accuracy makes sense.

The results become more confusing when we move to the next pair of columns, for F-score. In many cases the default beta levels lead to higher F-scores, despite lowering tag accuracy. The changes are not significant, but consider the BFGS 2000k model in which tag accuracy is decreasing by 0.91% and F-score increases by 0.11%. This fits with the theory that adaptation is occurring, since it suggests the words now being incorrectly tagged were going to be treated incorrectly by the parser anyway.

What all this discussion is leading up to is contained in the last two columns, for speed. The results in these columns follow the pattern in the column for ambiguity, which fits with the initial aim of this

	Tag Accuracy (%)	Ambiguity (cats / word)	Multi Tag Accuracy (%)		F-score (%)		Speed (sents / sec)	
Data	Default	Default	Default	Tuned	Default	Tuned	Default	Tuned
GIS								
0k	91.49	1.28	96.42	96.32	83.83	83.82	46.7	51.7
40k	91.62	1.24	96.16	96.36	84.07	84.05	58.9	56.9
400k	91.87	1.20	95.92	96.46	84.18	83.98	68.3	56.5
2000k	91.98	1.19	96.00	96.64	84.24	84.22	70.4	55.6
4000k	91.98	1.19	96.00	96.68	84.37	84.32	71.3	55.0
BFGS								
0k	91.38	1.27	96.30	96.29	83.73	83.73	49.3	52.1
40k	91.51	1.23	95.93	96.24	84.06	84.14	62.4	57.3
400k	91.46	1.12	94.63	96.12	83.68	83.89	95.2	60.4
2000k	91.95	1.16	95.61	96.52	84.42	84.31	81.4	57.8
4000k	91.97	1.18	95.76	96.55	84.30	84.19	78.9	57.2
AP								
0k	91.41	1.04	92.58	95.65	78.32	83.74	136.0	59.2
40k	91.42	1.03	92.42	95.77	78.75	84.04	161.0	64.0
400k	91.68	1.02	92.34	96.08	78.55	83.91	164.0	62.9
MIRA								
0k	91.42	1.31	96.42	96.19	83.64	83.69	43.0	50.6
40k	91.61	1.24	96.09	96.25	83.96	83.93	58.7	56.7
400k	91.78	1.16	95.43	96.39	83.96	83.92	77.4	57.5
2000k	91.83	1.13	95.26	96.60	83.96	84.11	85.1	56.8
4000k	91.93	1.12	95.09	96.62	83.99	84.10	90.2	55.0

TABLE 6.3: Performance comparison for models using default, or tuned beta levels.

work, to reduce ambiguity and thus improve speed. This demonstrates that while the decision to tune the beta levels ensures a fair comparison, it means we are not getting the most out of the models that we can. Also, the results in Table 6.3 for the Average Perceptron based models clearly indicate that a flexible strategy for beta level optimisation that is specific to each model is required. Also, it would be preferable to develop a strategy that does not require a great deal of gold standard data, as such resources are expensive to produce.

The first step to developing such a strategy is to explore the behaviour of the parser and the supertagger, something that has not been systematically done previously. In the next section I take the first steps towards a solution by exploring the behaviour of the parser on section 00 of the WSJ.

6.5 Aggregated Analysis of Parser Behaviour

I performed a series of tests to get some sense of the nature of the data and how the parser behaves as beta and the dictionary cutoff vary. I considered one hundred beta values spaced logarithmically between 1.0 and 0.0001, paired with twenty-two cutoffs between 1 and 211. For each pair I parsed all of the sentences in section 00 of the WSJ.

When considering the plots in the following sections note that the colour scales vary between sets of plots, to ensure an adequate range of colour is visible. Also remember that the default beta levels and dictionary cutoffs are as follows:

- 0.075, 20
- 0.030, 20
- 0.010, 20
- 0.005, 20
- 0.001, 150

6.5.1 All Sentences

The first set of plots, in Figure 6.1 consider the average results across all sentences in section 00. Ambiguity appears as expected, smoothly increasing as the beta level decreases, though it is interesting to note how little difference the tag dictionary cutoff makes, particularly at higher beta values. The coverage plot is also sensible, showing loss of coverage on either side. The decrease on the left is due to explosions as the number of tags assigned is too large. The decrease on the right is caused by sentences that the parser fails to find a spanning analysis for, as the range of options provided by the supertagger is too small.

The next three plots, precision, recall and F-score, all have the same scale for more convenient comparison. The F-score plot is dominated by the recall variations, which are caused by the loss of coverage. Based on these results it appears as if there is only a very limited band in which high performance is possible, but while this may be the case for a single beta level and dictionary cutoff, in fact the system has five such levels. This makes it difficult to draw conclusions from these plots, as a sentence that is not parsed for one pair of parameters may be parsed by another pair. It doesn't make sense to penalise

the first pair on accuracy for sentences that are not parsed at that level, as those sentences have not been completely knocked out.

The supertag accuracy plot is presumably dominated by the coverage issues as well, as the accuracy here is calculated based on the final set of tags the parser uses. Finally, the plot for parsing time follows the same pattern as the ambiguity plot. This makes sense as the higher the ambiguity the larger the number of possible derivations the parser must consider.

6.5.2 Only Successfully Parsed Sentences

If we average over only the sentences that were parsed at each point we remove the dominating effect of coverage, producing the plots in Figure 6.2. No plot is included for coverage, as we are only considering sentences that are parsed. The trends in the ambiguity and parsing time plots do not change significantly, but the maximum time is 25% lower, and is not approached until further to the right. This is due to the removal of the times for sentences that exploded. It may not be surprising, but it does demonstrate the cost of exploding.

The next three plots, for accuracy, recall and F-score, change considerably. Note the extreme change in scale, from a lower end of 0.72 in the previous set of plots to 0.82 here. Interestingly there is still a slight decrease in recall on the far right, but overall the plot is much more similar to the precision plot.

An interesting pattern that can be seen in the previous set of plots as well as here is the drop in performance when decreasing the tag dictionary cutoff from 11 to 1. This is presumably just the result of noise in the data, which is lost once the cutoff is actually in use.

While these plots avoid the coverage issue of the previous set, they introduce a new issue – no two points on the plot are actually directly comparable. The sentences being used in the calculations at one point are not the same as those used at another point. This is a serious issue because the sentences being parsed further to the right are generally shorter, easier sentences, which explains why accuracy is so high.

6.5.3 Only Sentences Always Parsed

It turns out that $1,318$ of the $1,913$ sentences in section 00 are parsed at all points in these plots. So only using sentences that are always parsed is an easy way to avoid the coverage issue, and the need to have the same sentences used for every measurement.

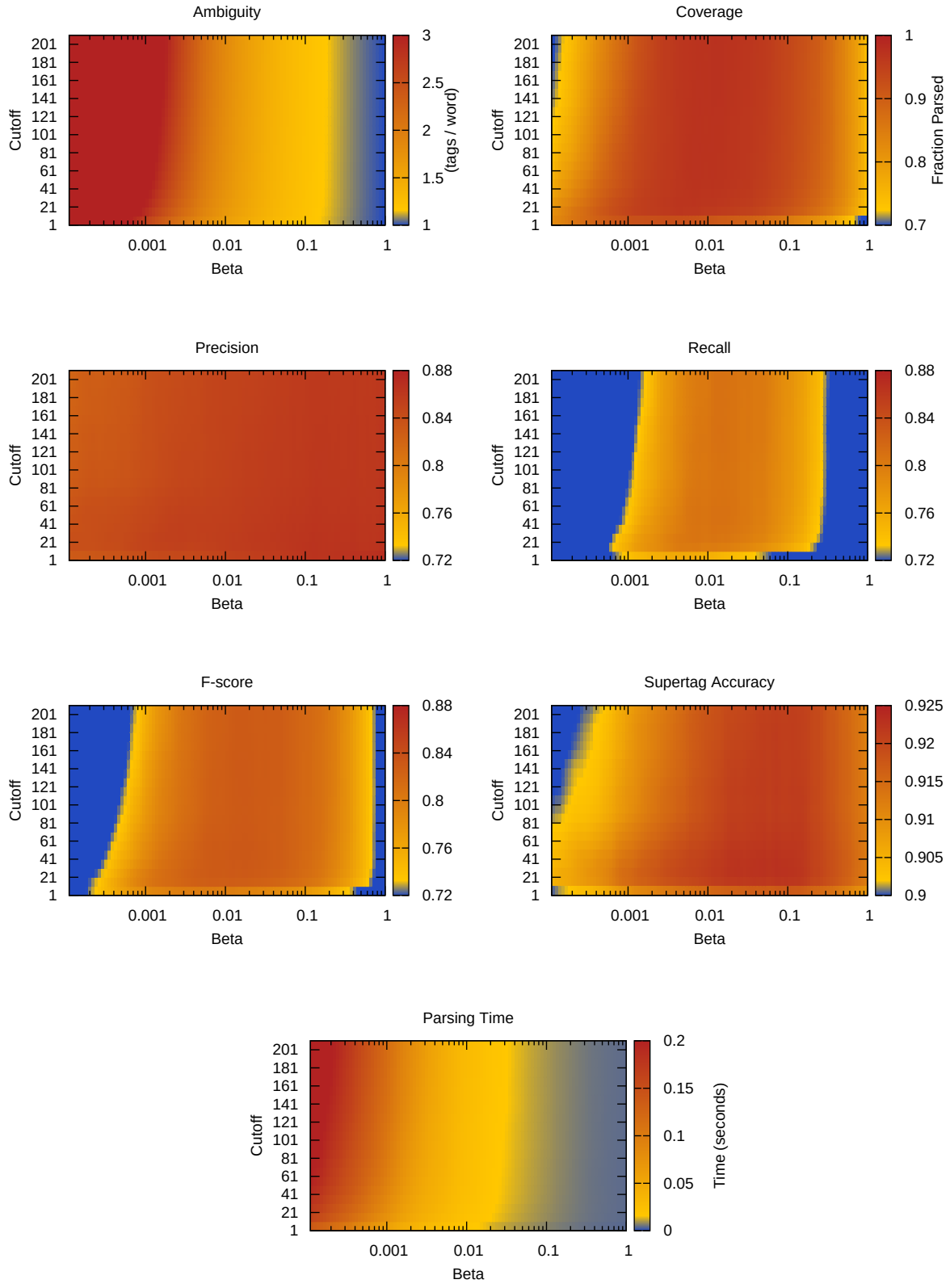


FIGURE 6.1: Parsing behaviour over all sentences in section 00 of the WSJ.

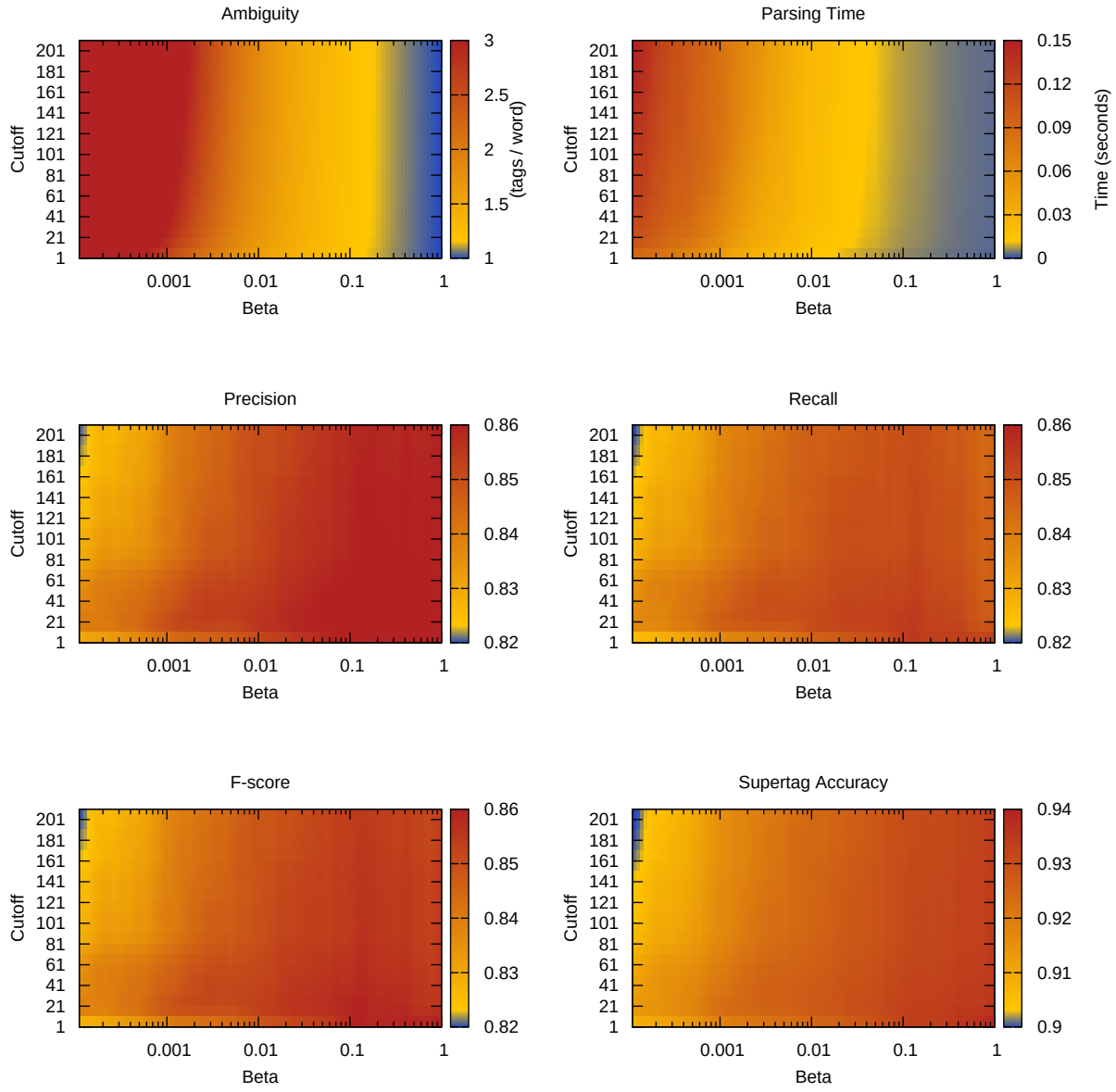


FIGURE 6.2: Parsing behaviour over parsed sentences in section 00 of the WSJ.

The changes observed between the two previous sets of plots continue here. Now accuracy is varying between 85% and 91%, compared to 82% and 86% in the previous set. Parsing time is only extending up to 0.13 seconds, rather than 0.15. And supertag accuracy has increased as well. No major changes in the patterns are observed, with the exception of parsing time, where the increase is concentrated further to the left of the plot. This is probably a result of the complexity of the sentences being excluded from this plot, and is explored further in the following section.

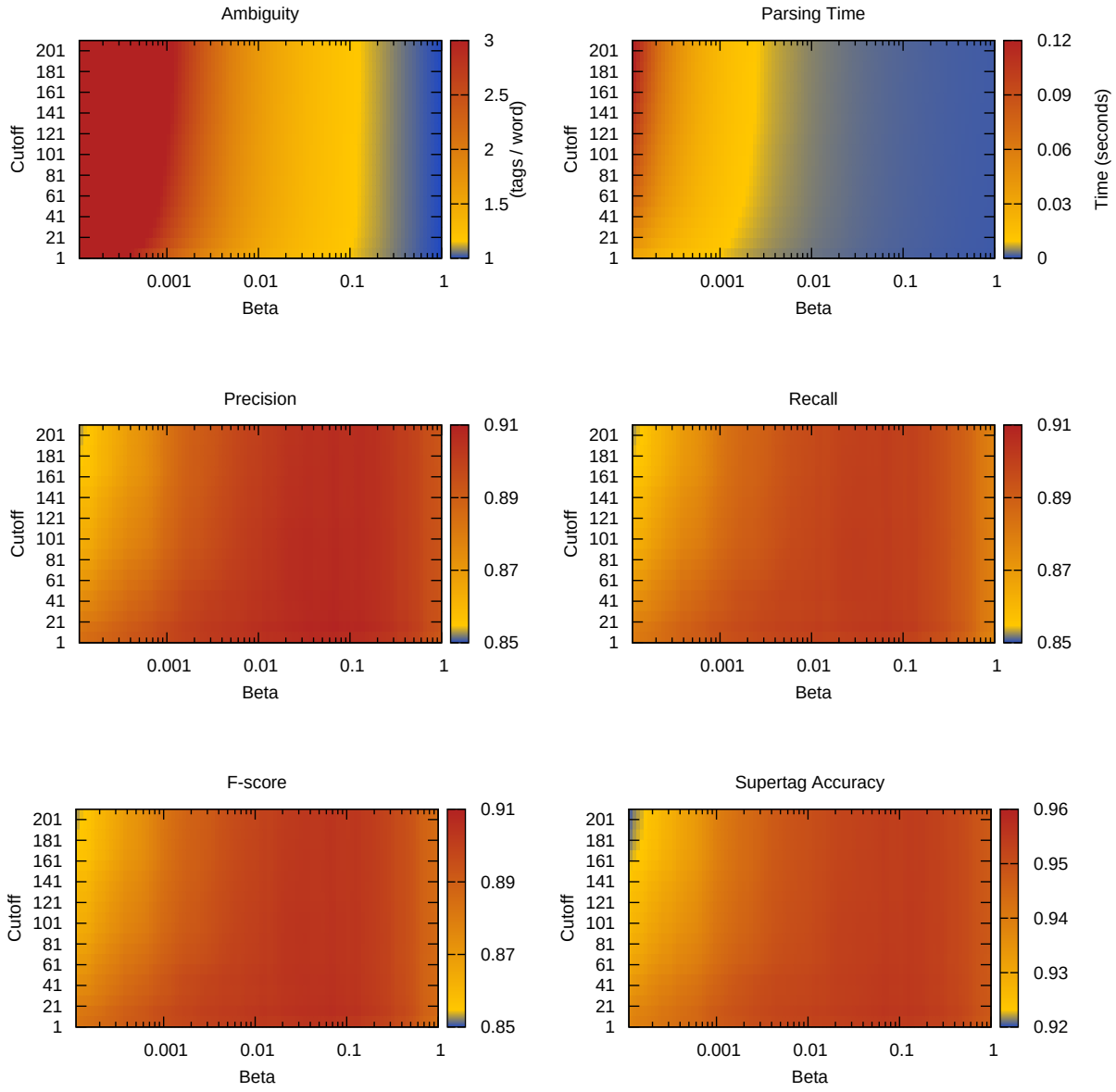


FIGURE 6.3: Average parsing behaviour for parsed sentences in section 00 of the WSJ.

While the sentences these plots are based on make up more than two thirds of the set, they are not actually our primary concern when choosing parameters. They are important for the choice of the first level, but irrelevant after that, since all of these sentences are guaranteed to be parsed at the first level.

Min Length	Max Length	Total	Always Parsed
1	12	288	287
13	17	314	310
18	21	289	283
22	26	321	258
27	33	361	160
34	250	340	20

TABLE 6.4: Break down of data based on sentence length.

6.6 Behaviour by Sentence Length

Parsing efficiency is heavily dependent on sentence length. Longer sentences are generally more complex and therefore more time consuming and difficult to parse. At the moment the parser functions exactly the same for longer sentences as it does for shorter sentences. By using different parameters depending on the length of the sentence we may be able to prevent explosions for larger sentences.

Table 6.4 summarises the six sets the sentences were divided into. These ranges were chosen to balance the size of the sets as much as possible. This table alone makes it clear that different strategies may be needed depending on sentence length. Almost all of the sentences containing less than twenty two words are parsed regardless of the parameters used; less than half of the sentences that are twenty two words or longer have such flexibility.

As before, more than one set of plots have been included. The first set provides the results for all sentences, while the second set are based on only sentences that always receive a parse. The second set considered in the previous section, calculated based on all the sentences parsed at each point, has not been included for the reason raised in the previous section – it is difficult to draw meaningful conclusions when the set of sentences being used is varying across the graph.

Also note that the plots have been ordered by row from left to right. This means the top left plot is for sentences of length 1 to 12, the top middle plot is for sentences of length 13 to 17, the top right plot is for sentences of length 18 to 21 and so on.

6.6.1 All Sentences

Not all of the metrics have been included either. Ambiguity followed the same trends in all cases, and precision also displayed very similar trends, though there was an overall decrease in precision as

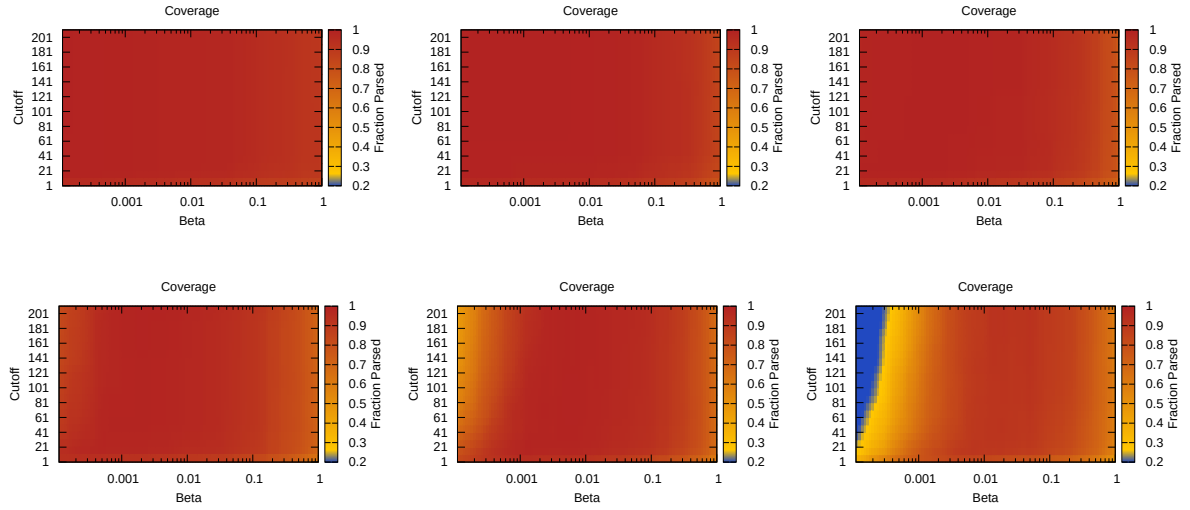


FIGURE 6.4: Stats by length for coverage, over all sentences.

sentence length increased. Finally, plots of F-score have not been included as they are dominated by recall and add little extra information.

The breakdown for coverage clearly demonstrates that the parser behaves quite differently depending on sentence length. For the first three plots there is almost no decrease in coverage on the left hand side, and only a slight decrease on the right. This indicates that for shorter sentences the parser's chart rarely explodes and so the only problem for coverage is when the tag set is too restrictive and so no spanning analysis can be formed.

Meanwhile, longer sentences experience major issues as the tag set grows. This makes sense, as a slight increase in tag set size across a long sentence will lead to many new potential derivations. This decrease in coverage for smaller beta values is important to remember for the following analysis, particularly for the longest sentences, as it will decrease performance on all other metrics.

The next set of plots, for accuracy of the final supertag set used by the parser, are slightly surprising. As in the previous section, we observe a decrease in accuracy as we move left from the centre of the plot, suggesting that given more options, the parser is less likely to make the right choice. Interestingly the parser is better at the medium length sentences than the shortest group, particularly for high tag dictionary cutoffs and low beta levels, where the lowest result of all the plots is found. As expected, results for the longest sentences are noticeably poorer throughout.

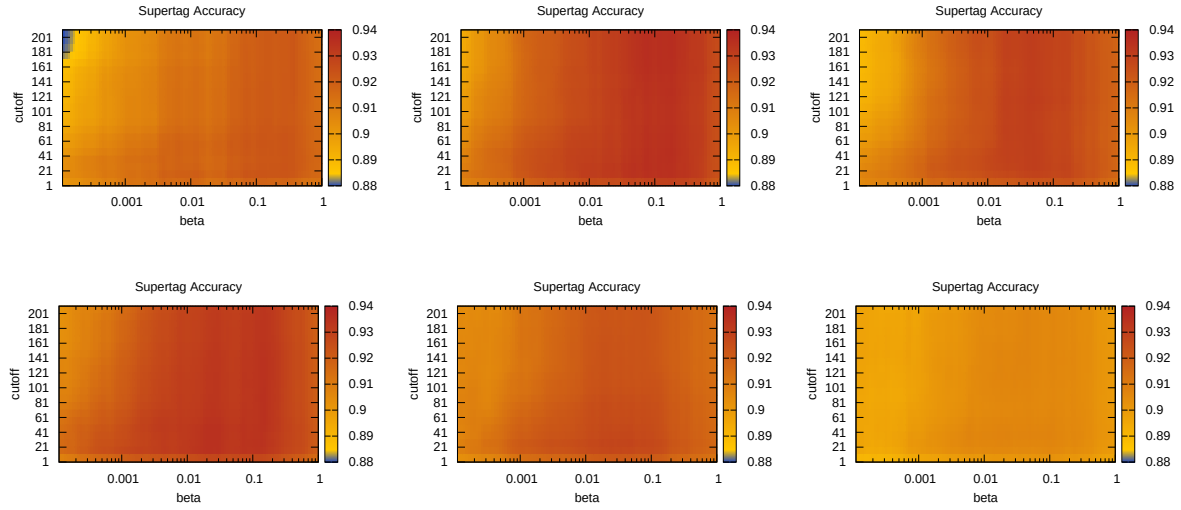


FIGURE 6.5: Stats by length for supertag accuracy, over all sentences.

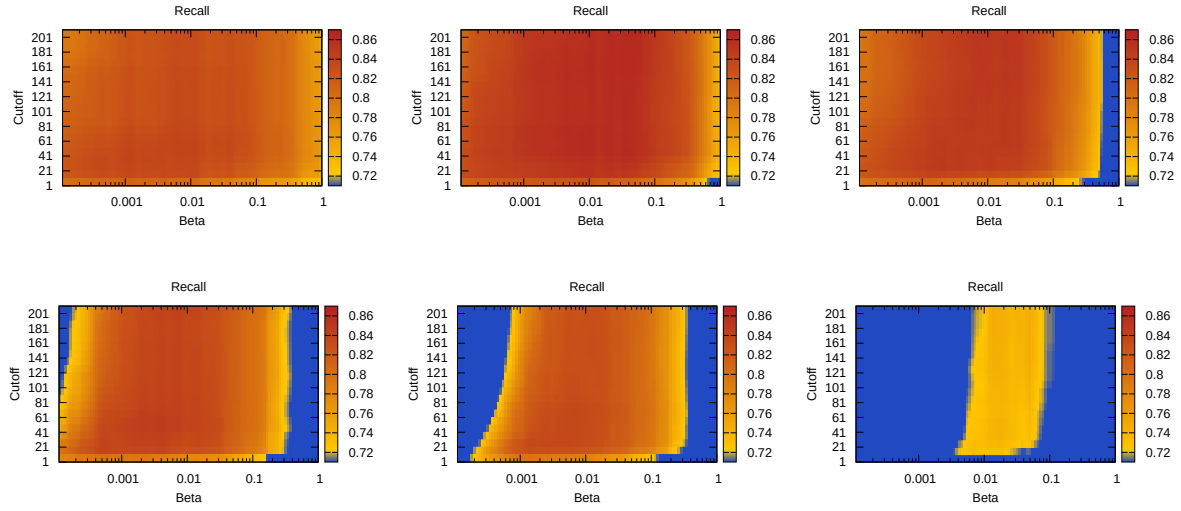


FIGURE 6.6: Stats by length for recall, over all sentences.

The results for recall largely mirror the observations in the previous section. For the shorter sentences, where coverage is not a major issue, the plots generally reflect the trends in supertag accuracy. For longer sentences the plots are dominated by the sentences that are only parsed in a narrow range outside of which either explosions occur, or no spanning analysis can be found.

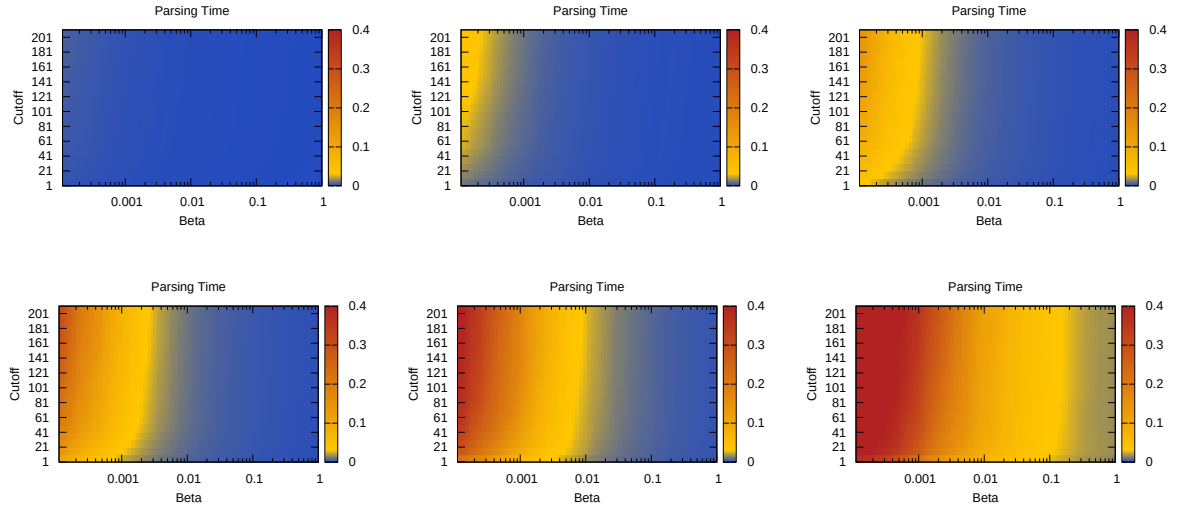


FIGURE 6.7: Stats by length for speed, over all sentences.

Finally, the results for speed are as expected. The longer a sentence is the longer it takes to parse. It is worth noting the particularly bad performance for the longest sentences, which are slow to parse even at the highest beta levels.

6.6.2 Only Sentences Always Parsed

As for the previous set of results, ambiguity plots for sentences that are always parsed are not included as they demonstrate the same trends as previously. Also, the speed results are very similar to those in the previous section, with the exception that the penalty for longer sentences is no longer as bad.

The most significant difference between the supertag accuracy plots in Figure 6.8 and those in Figure 6.5 is the improvement for longer sentences. This makes sense, as the sentences that are not always parsed are the hard ones, and ignoring them should provide a boost to performance. It is slightly surprising how great a difference it makes, since these plots show accuracy for the longer sentences as actually better than for shorter sentences.

Plots for precision have been included in this section because there is a lot more going on. The impressive decrease in performance in the top right plot indicates again that the parser is more likely to make the wrong decision when given more choice. The most likely reason for this trend being less visible in the second row of plots, for longer sentences, is that the parser's chart explodes at that point and those sentences are ignored for these plots. There is some particularly strange behaviour in the bottom left

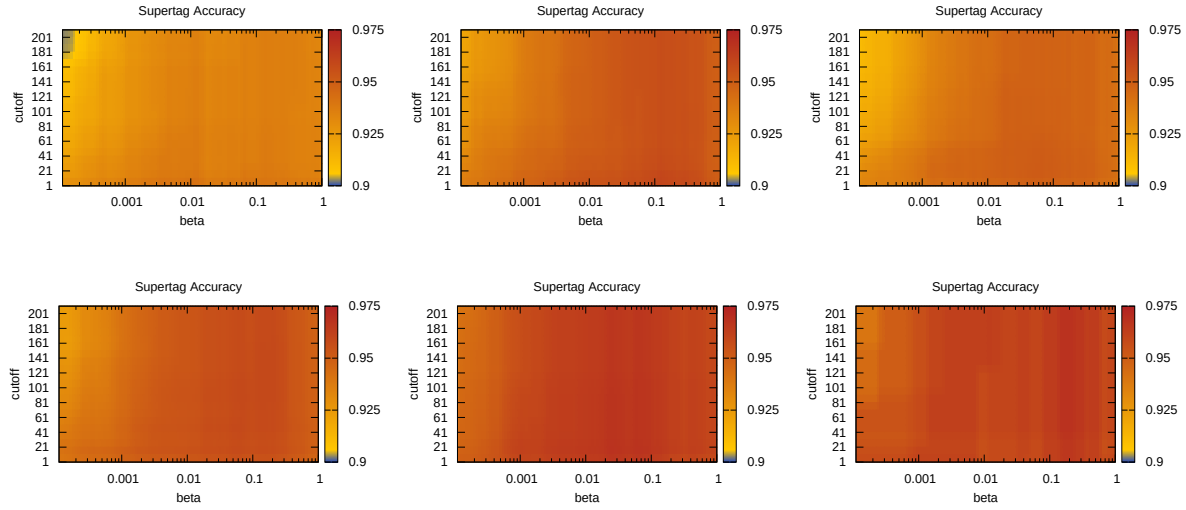


FIGURE 6.8: Stats by length for supertag accuracy, over parsed sentences.

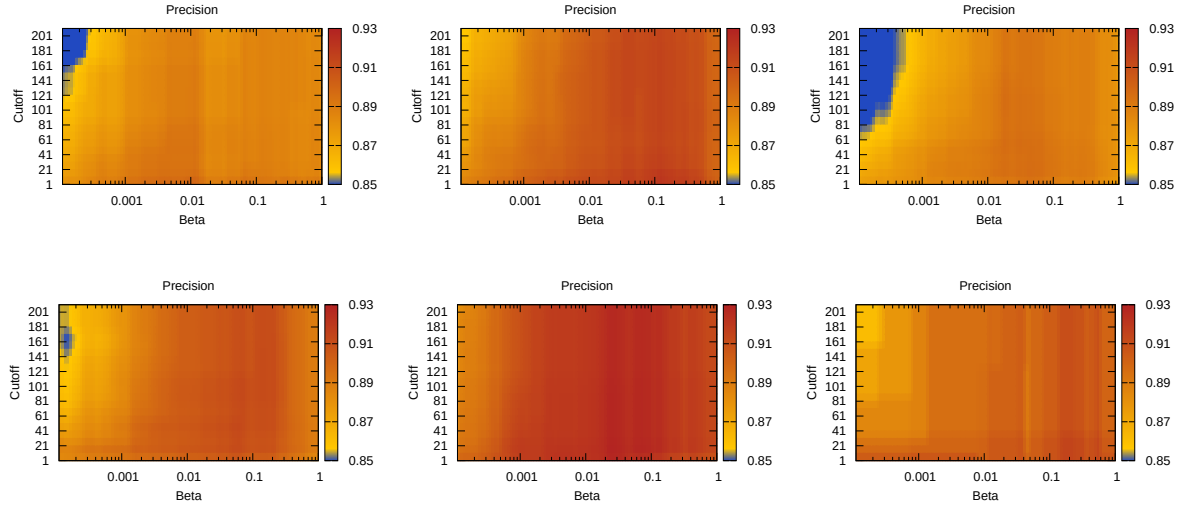


FIGURE 6.9: Stats by length for precision, over parsed sentences.

plot, where at the lower beta levels precision drops as the cutoff is raised, but then improves again. The reason for this is unknown, though it could be a quirk of the sentences being used, since for the lower row a considerable number of sentences are being ignored.

Now that the sentences that are not always parsed are ignored we can see very different trends for recall. Instead we observe patterns similar to those in the precision plots and no other particularly strange behaviour. Since the precision and recall plots are so similar, plots of F-score have not been included.

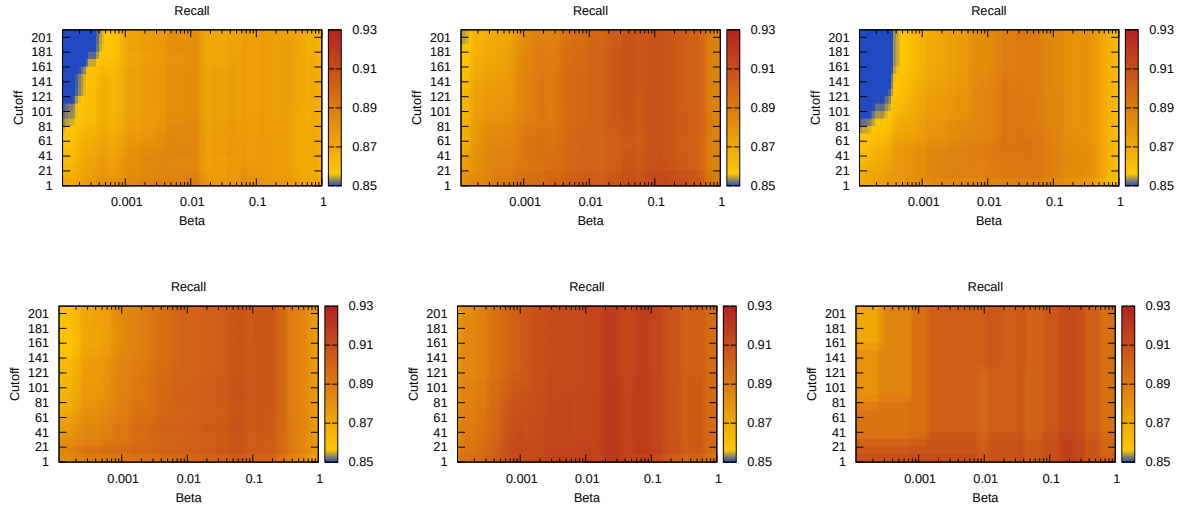


FIGURE 6.10: Stats by length for recall, over parsed sentences.

6.7 Interesting Sentences

The exploration so far has focused on aggregated behaviour across larger sets of sentences. The problem with this approach is that it is dominated by the common cases, in particular the sentences that are parsed regardless of the parameters used. In this section I consider some specific examples to explore the factors affecting accuracy.

6.7.1 Sentence 12

Before considering the exceptional cases it is worth considering one of the common cases – a sentence that is parsed for all the parameter pairs considered. The sentence I have chosen is:

```
We have no useful information on whether users are at risk
, said James A. Talcott of Boston 's Dana-Farber Cancer
Institute .
```

The gold standard POS tags assigned to each word in CCGBank are:

```
We|PRP have|VBP no|DT useful|JJ information|NN on|IN
whether|IN users|NNS are|VBP at|IN risk|NN ,|, said|VBD
James|NNP A.|NNP Talcott|NNP of|IN Boston|NNP 's|POS
Dana-Farber|NNP Cancer|NNP Institute|NNP .|.
```

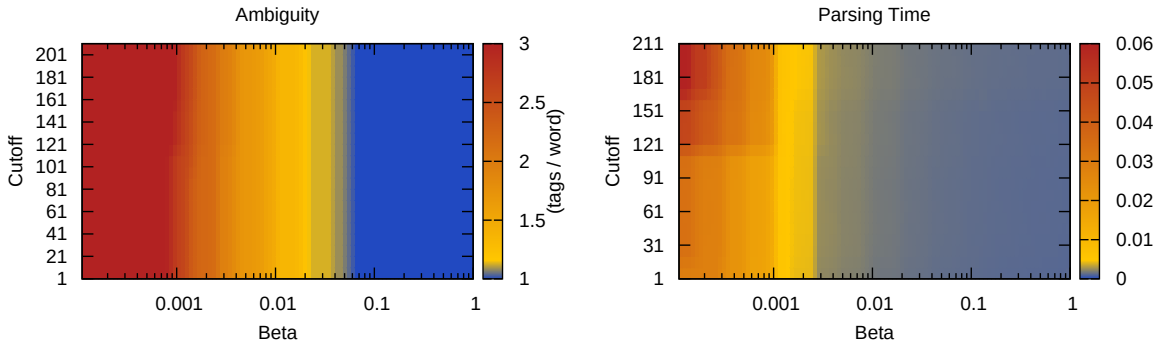


FIGURE 6.11: Parsing behaviour for the sentence 12 sentence in section 00.

In this example there were no errors made by the C&C POS tagger, as is the case for slightly more than half the sentences in section 00. The gold standard supertags are:

```
We|PRP|NP have|VBP|(S[dcl]\ NP)/NP no|DT|NP[nb]/N
useful|JJ|N/N information|NN|N on|IN|(NP\ NP)/S[qem]
whether|IN|S[qem]/S[dcl] users|NNS|N are|VBP|(S[dcl]\ NP)/PP
at|IN|PP/NP risk|NN|N ,|,|, said|VBD|(S[dcl]\ S[dcl])/NP
James|NNP|N/N A.|NNP|N/N Talcott|NNP|N of|IN|(NP\ NP)/NP
Boston|NNP|N 's|POS|(NP[nb]/N)\ NP Dana-Farber|NNP|N/N
Cancer|NNP|N/N Institute|NNP|N .|.|.
```

Since this example is parsed everywhere and in the same way, the coverage, F-score, precision, recall and supertag accuracy plots have not been included. For all of these metrics this sentence had a perfect score.

The ambiguity plot appears very similar to the average plots considered in the previous sections. Tag dictionary cutoff appears to have very little effect, while decreasing the beta value smoothly increases the ambiguity. Also, as observed previously, the parsing time plot follows the same patterns as the ambiguity plot.

6.7.2 Sentence 577

This sentence demonstrates the power of the POS tagger. The sentence is:

```
One claims he 's pro-choice .
```

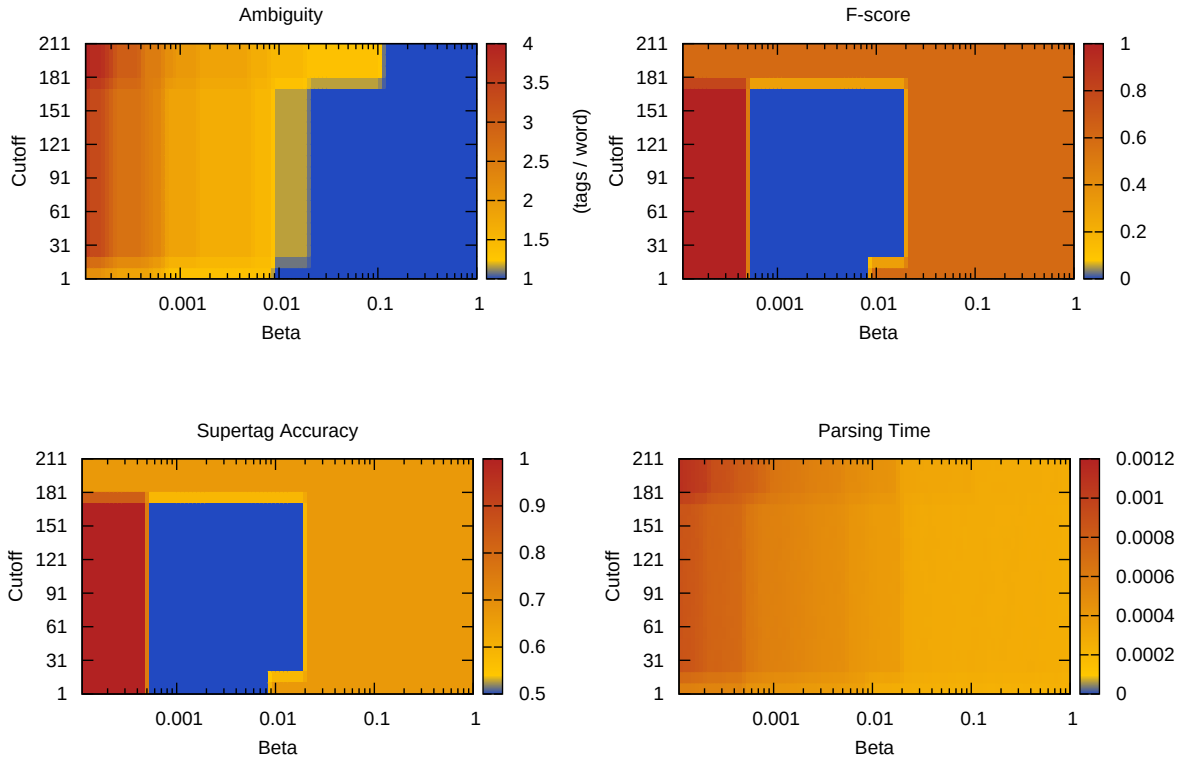


FIGURE 6.12: Parsing behaviour for the sentence 577 sentence in section 00.

The gold standard POS tags are:

```
One|NN claims|VBZ he|PRP 's|VBZ pro-choice|JJ .|.
```

The C&C POS tagger makes two mistakes:

- One, NN to CD
- claims, VBZ to NNS

By misinterpreting the word `One` as a number, rather than a noun, the POS tagger has thrown a serious spanner in the works. The mistake is understandable, as `One` would be a number in most sentences.

This sentence is parsed for all pairs of parameters tested, but as Figure 6.12 shows, the behaviour is slightly odd.

The gold standard supertags and labelled dependencies are:

```
 One|NN|N
claims|VBZ|(S[dcl]\ NP)/S[dcl]
he|PRP|NP
’s|VBZ|(S[dcl]\ NP)/(S[adj]\ NP)
pro-choice|JJ|S[adj]\ NP
.|.|.

claims_2 (S[dcl]\ NP)/S[dcl] 2 ’s_4
claims_2 (S[dcl]\ NP)/S[dcl] 1 One_1
’s_4 (S[dcl]\ NP)/(S[adj]\ NP) 1 he_3
’s_4 (S[dcl]\ NP)/(S[adj]\ NP) 2 pro-choice_5
pro-choice_5 S[adj]\ NP 1 he_3
```

In the region furthest to the right the supertagger only supplies one tag per word. The mistakes made by the POS tagger lead it to treat `One claims` as a noun, where `One` is modifying `claims`. With the last four tags correct the parser is able to identify some of the correct dependencies, as shown below.

```
 One|CD|N/N
claims|NNS|N
he|PRP|NP
’s|VBZ|(S[dcl]\ NP)/(S[adj]\ NP)
pro-choice|JJ|S[adj]\ NP
.|.|.

One_1 (N/N) 1 claims_2
’s_4 (NP\ NP) 1 claims_2
’s_4 ((S[dcl]\ NP)/(S[adj]\ NP)) 1 he_3
’s_4 ((S[dcl]\ NP)/(S[adj]\ NP)) 2 pro-choice_5
pro-choice_5 (S[adj]\ NP) 1 he_3
```

In the middle region, where F-score falls dramatically, the supertagger has the flexibility to assign extra tags:

```
 One - N/N, (S/S)/(S/S)
claims - N
he - NP
’s - (S[dcl]\ NP)/(S[adj]\ NP), (S[dcl]\ NP)/NP
pro-choice - S[adj]\ NP, (S\ NP)
(S\ NP), N, (S[adj]\ NP)/S[dcl]
.  - .
```

However, this simply misleads the parser, which chooses a derivation using one more incorrect tag than previously:

```
 One|CD|N/N
claims|NNS|N
he|PRP|NP
’s|VBZ|(S[dcl]\ NP)/NP
pro-choice|JJ|S[adj]\ NP
.|.|.

One_1 (N/N) 1 claims_2
’s_4 ((S[dcl]\ NP)/NP) 1 he_3
’s_4 ((S[dcl]\ NP)/NP) 2 claims_2
pro-choice_5 (S[adj]\ NP) 1 claims_2
```

In the left-most region the supertagger finally has enough flexibility to give all of the correct tags as options, which the parser is able to combine into the correct derivation. Importantly, tests where the parser was given the correct supertags led to the correct derivation, and when the supertagger was given the correct POS tags it produced the right supertags.

Clearly an improvement at either of the earlier stages will lead to an improvement in parser performance. The other important feature of this example is that providing more supertags is not always best – if they are not the correct supertags they may simply mislead the parser.

6.7.3 Sentence 1791

This sentence demonstrates interesting behaviour by the parser, and provides one possible explanation for the accuracy improvements described in the previous chapter's results. The sentence is:

```
If the Japanese companies are seriously considering their
survival , they could do at least three things to improve the
situation :  raise salaries higher than those of financial
institutions ; improve working conditions -LRB- better offices
and more vacations , for example -RRB- ; accept and hire more
labor from outside Japan .
```

With gold standard POS tags it is:

```
If|IN the|DT Japanese|JJ companies|NNS are|VBP seriously|RB
considering|VBG their|PRP$ survival|NN ,|, they|PRP could|MD
do|VB at|IN least|JJS three|CD things|NNS to|TO improve|VB
the|DT situation|NN :|:  raise|VB salaries|NNS higher|JJR
than|IN those|DT of|IN financial|JJ institutions|NNS ;|;
improve|VB working|NN conditions|NNS -LRB-|LRB better|JJR
offices|NNS and|CC more|JJR vacations|NNS ,|, for|IN
example|NN -RRB-|RRB ;|; accept|VB and|CC hire|VB more|JJR
labor|NN from|IN outside|JJ Japan|NNP .|.
```

Our POS tagger produces the same results for all words except the following:

- :, : – IN
- working, NN – VBG
- -LRB-, LRB – JJ
- -RRB-, RRB – NNP
- outside, JJ – IN

Interestingly, Figure 6.13 shows that the only times this sentences receives a parse are when the supertags chosen by the parser are incorrect. At all other times either an explosion occurs or no spanning analysis can be found. In this case if the supertagger supplies the correct tags we will fail to find an analysis, but if it had been trained to supply the tags that the parser wanted, we would find an analysis. Cases like this could be the reason for the observed accuracy improvements during the adaptive training experiments.

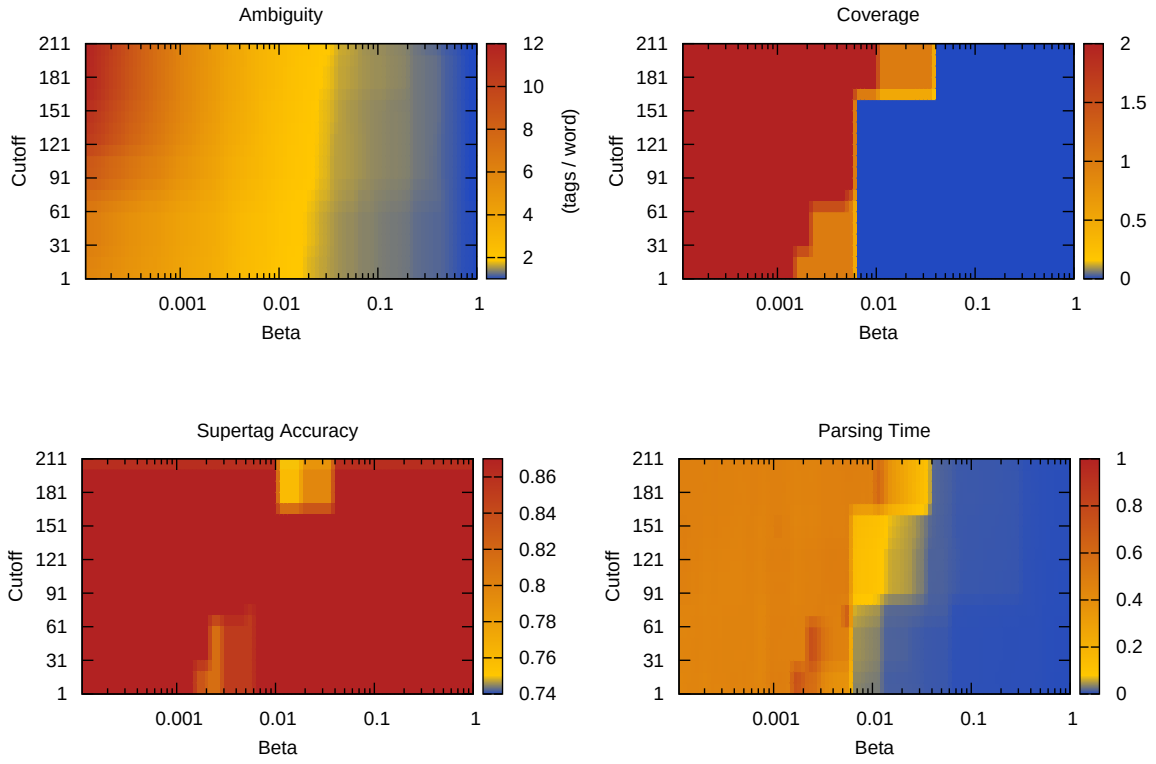


FIGURE 6.13: Parsing behaviour for the 1791^{st} sentence in section 00.

6.7.4 Sentence 212

In the sentence:

```
The U.S. , claiming some success in its trade diplomacy
, removed South Korea , Taiwan and Saudi Arabia from a
list of countries it is closely watching for allegedly
failing to honor U.S. patents , copyrights and other
intellectual-property rights .
```

Despite no POS tagging errors, we observe strange patterns in supertag accuracy. In the area of low F-score to the left in Figure 6.14 the parser is experiencing an explosion, and in the area to the right no spanning analysis is found. This is strange as those are not the areas where the supertags chosen by the parser are least accurate. Meanwhile the point of lowest supertag accuracy and the line of low accuracy above it does not prevent a parse from being found.

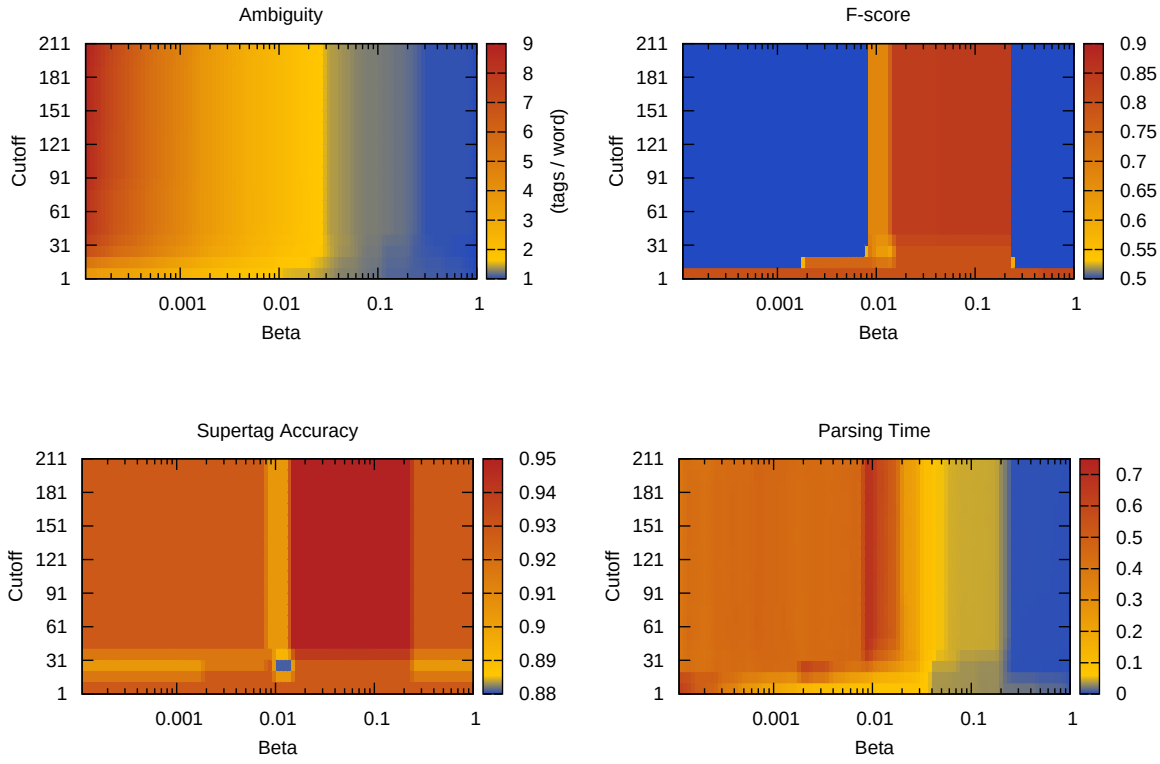


FIGURE 6.14: Parsing behaviour for the sentence 212 sentence in section 00.

The speed plot also demonstrates strange behaviour. The thin band of particularly slow processing lines up with the edge of the region in which a parse is found. This is probably a peak because after this the explosion occurs early enough for time to be saved, but it is still odd.

6.7.5 Sentence 274

Despite being extremely short and no POS tag errors occurring, the following sentence displays some surprising behaviour:

```
Previously , watch imports were denied such duty-free
treatment .
```

In Figure 6.15 the blue areas are caused by the parser failing to find a spanning analysis. What makes this so strange is that the supertag accuracy is the same in the middle area as it is in the top right, where the parser fails to find a spanning analysis. Also, the speed varies strangely across the bottom of the plot, increasing more rapidly for slightly higher cutoffs, despite no noticeable change in ambiguity.

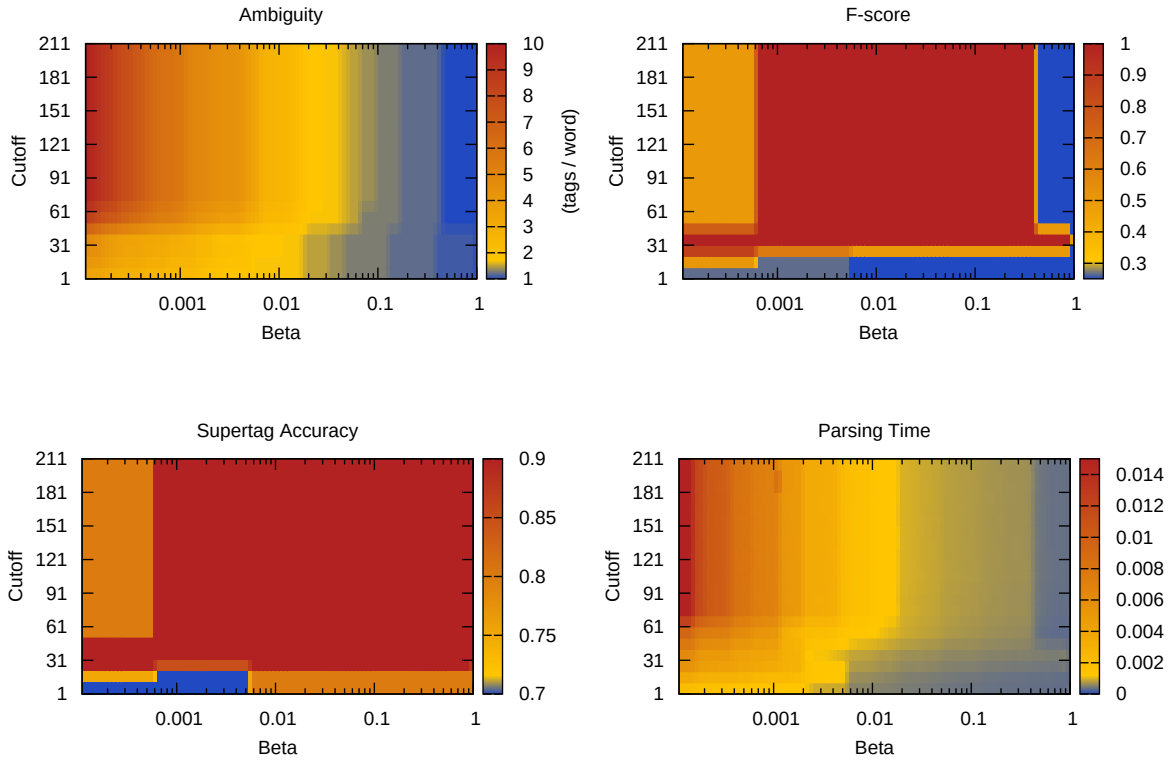


FIGURE 6.15: Parsing behaviour for the sentence 274 sentence in section 00.

6.7.6 Sentence 302

This sentence is an example of a more complex case of the sentence 577:

```
First , they are designed to eliminate the risk of prepayment
- mortgage-backed securities can be retired early if interest
rates decline , and such prepayment forces investors to
redeploy their money at lower rates .
```

The POS tagger mislabels `forces` as a noun instead of as a verb, and `First` as an adverb instead of as an adjective. The strange behaviour shown in Figure 6.16 is a result of the supertagger gradually providing more tags and the parser doing gymnastics to get the correct derivation. Being longer than sentence 577 there is more scope for variation, as we see in the plot for F-score, which has many regions, bounded vertically and horizontally by points at which the supertagger is able to change the tag set.

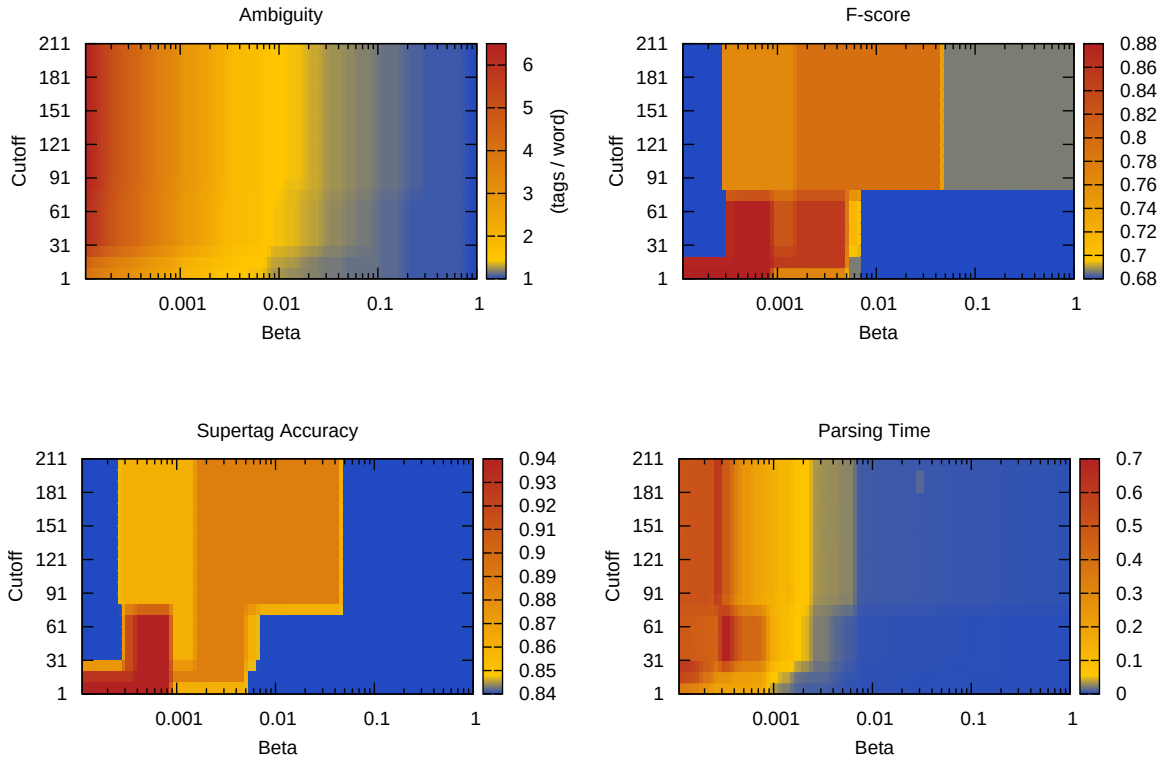


FIGURE 6.16: Parsing behaviour for the sentence 302 sentence in section 00.

6.7.7 Sentence 596

This sentence demonstrates similar strange behaviour as sentence 212:

```
That commercial - which said Mr.  Coleman wanted to take
away the right of abortion even in cases of rape and incest
, a charge Mr.  Coleman denies - changed the dynamics of
the campaign , transforming it , at least in part , into a
referendum on abortion .
```

In Figure 6.17 in the left area the parser experiences an explosion, and in the right area no spanning analysis is found. Strangely, the region in which supertag accuracy is lowest is in neither of these regions and does not actually seem to influence F-score at all. We also observe that speed increases considerably towards the edges of the region in which a parse is found, then drops away and remains fairly constant in the explosion area.

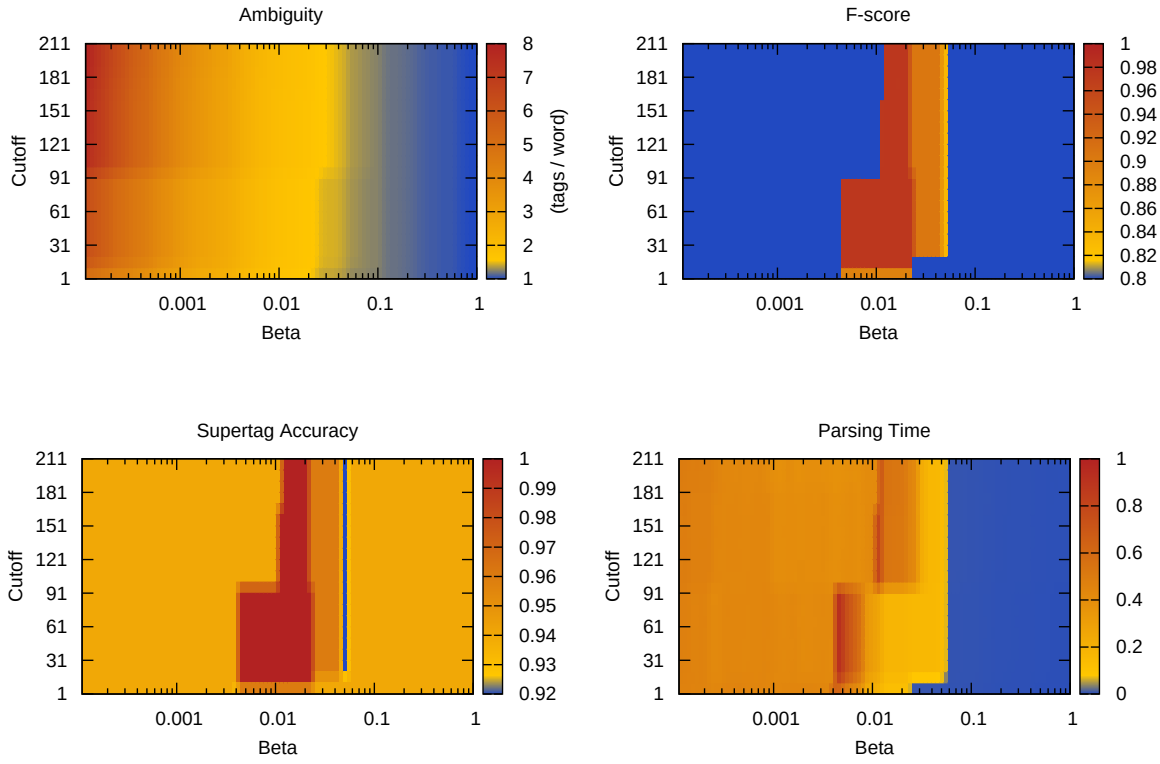


FIGURE 6.17: Parsing behaviour for the sentence 596 sentence in section 00.

6.7.8 Sentence 691

The final sentence I consider is one of the most strange:

> `In her wake she left the bitterness and anger of a principal who was her friend and now calls her a betrayer ; of colleagues who say she brought them shame ; of students and parents who defended her and insist she was treated harshly ; and of school-district officials stunned that despite the bald-faced nature of her actions , she became something of a local martyr .`

A single POS tag error is made – `stunned` is labelled as a verb instead of as an adjective. Across the very bottom and in the top right area no spanning analysis is found, while in the left area explosions occur.

In all the other areas we get a variety of strange patterns. The precision and recall above a tag dictionary cutoff of 31 appear to be the reverse of the supertag accuracy. The thin column of higher recall and precision is where supertag accuracy is lowest and in the regions where supertag accuracy is highest the parser fails either because of explosions or not being able to form a spanning analysis.

Speed also exhibits strange behaviour, following the trends in the F-score plot more than in the ambiguity plot as would be expected. In particular, the time taken increases much sooner below a tag dictionary cutoff of 31.

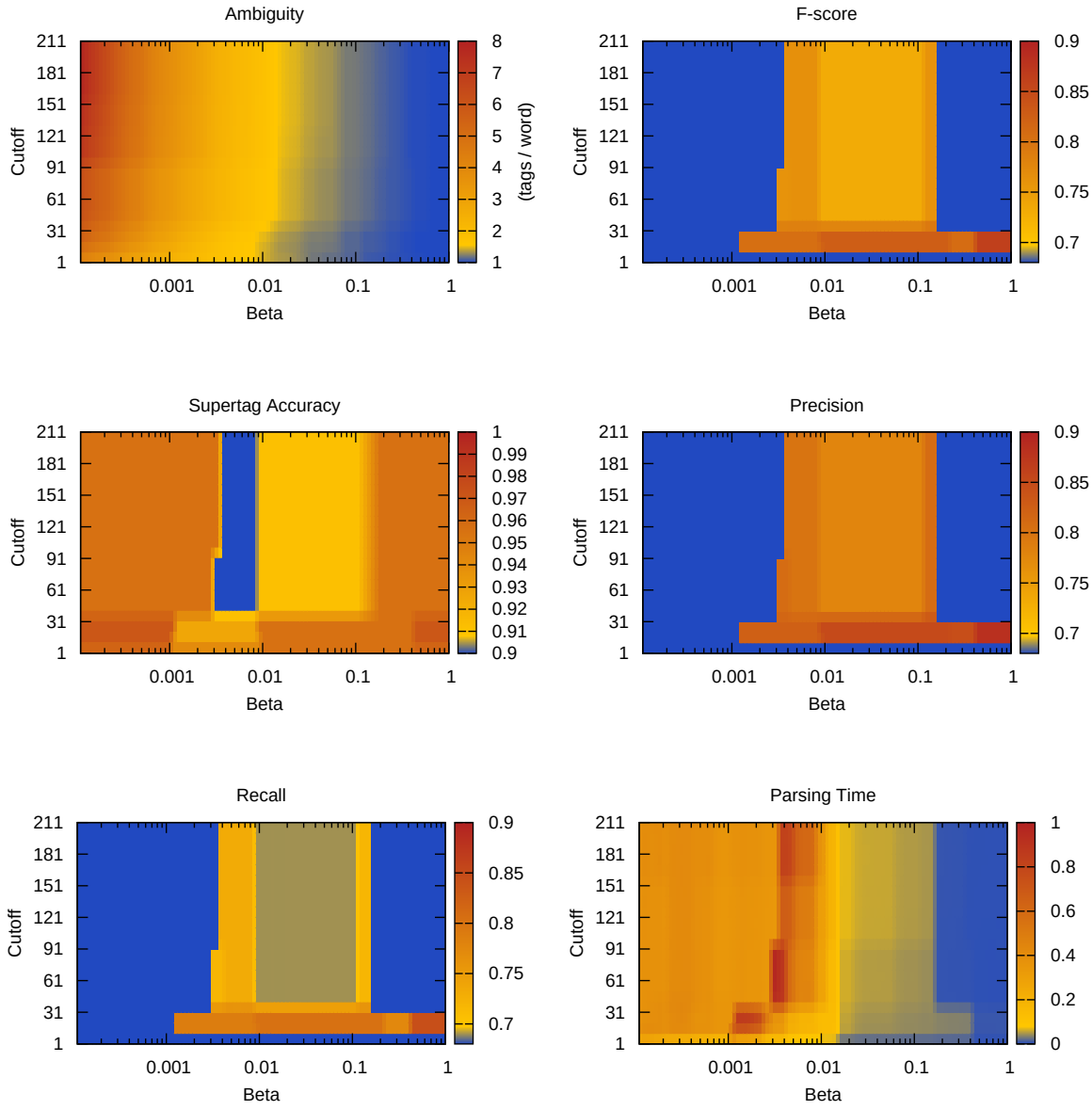


FIGURE 6.18: Parsing behaviour for the sentence 691 sentence in section 00.

6.8 Optimal Coverage Algorithm

Initially I focused on choosing beta levels that would maximise speed while achieving full coverage. Using a set of sentences representative of the target corpus, the algorithm translates each sentence into a line segment, and then locates a set of points such that every line segment contains a point.

To translate each sentence into a line segment involves two binary searches on beta levels. The first is for the largest beta value before a spanning analysis is not found. The second is for the last beta value before the number of tags assigned leads to a chart explosion. These values define a line segment over which the sentence is successfully parsed.

Now our problem is essentially to determine a set of points that together allow all of the sentences to be parsed. We would also like these values to be as large as possible, as it is expected that larger values will lead to a speed improvement. Both of these goals can be achieved by the following greedy algorithm:

- Initially label all line segments as uncovered
- Create an empty list $next$
- For each of the endpoints in all of the lines, in sorted increasing order:
  - If it is the point at which the chart explodes, add it to $next$
  - Otherwise, if it is uncovered, set this value as a beta level, and label all of the line segments in $next$ as covered

6.8.1 Correctness

To show that the algorithm is correct two points must be proved: first that every line segment contains a point, and second that the set of points chosen are optimal.

Every line segment is added to the $next$ list at some point, since we iterate through all endpoints, and for the first endpoint of each segment we place the segment in the list. When a beta level is chosen it covers all sentences in the $next$ list, since they can be parsed at this level. Also, if there are any elements in the $next$ list there must be an uncovered endpoint still to be considered, since the segments in the $next$ list are all uncovered and only one of their endpoints have been seen. Therefore every segment will be placed in the $next$ list at some point, and every segment placed in the $next$ list will be covered.

To prove optimality consider the order in which the algorithm generates points. Every line segment must contain a point, therefore the line segment that has the smallest starting beta value creates a limit on the lowest beta level. It cannot be any higher than that value, otherwise the sentence that line segment corresponds to would not be parsed. Therefore we must place a beta level at that value, which the algorithm does. All other sentences that can be parsed at this point can now be ignored, since they will definitely be parsed. Now we have the same problem as initially, but on a smaller set. Repeated application will lead to all segments containing a point, and none of the points can be removed, or moved any higher without causing a sentence to not be parsed. Therefore the set of points chosen is optimal

6.8.2 Results

Using this algorithm and the results of the analysis from the previous section I performed a series of tests. Since this algorithm does not consider tag dictionary cutoff I ran separate tests at each of the cutoffs used in the tests above. Since all of the behaviour tests were performed with the baseline model, these tests were also performed with the baseline.

Table 6.5 clearly shows that this method is not perfect, but does produce fairly effective beta levels. With the exception of the first cutoff, coverage is at least 97.8% and most of the tests produce similar coverage to the baseline model. Interestingly the tests with more levels perform considerably better and are the only cases that out-perform the baseline. The best of these parameter sets does perform slightly worse, as shown in Table 6.6, but makes even more impressive speed gains on Wikipedia, while maintaining coverage above 98.5%.

6.8.3 Issues

Based on these results it seems that coverage optimisation algorithm is working, providing a speed boost while maintaining high coverage. However, we are not actually getting quite what we want. The main issue is that complete coverage is actually too weak a requirement – we want to get the right parse, not just any parse. Also, while the method will guarantee coverage for the sentences used, it is unclear how large the set needs to be to ensure good generalisation to an entire corpus. The other potential factor, optimising the dictionary cutoff does not appear to be a major issue. In general it appears the tag dictionary cutoff has very little influence on speed or accuracy, with the exception of extremely low and high values, which exhibit irregular behaviour. This is useful because it means we can effectively ignore the cutoff values and optimise the beta values alone.

	Betas					Number Parsed					Failed	WSJ Speed
Cutoff	1	2	3	4	5	1	2	3	4	5		(sents / sec)
Baseline	0.075	0.03	0.01	0.005	0.001	9042	282	262	99	167	148	48.5
1	0.25	0.017	0.0058	0.0010	0.00012	7195	1202	314	286	171	832	40.4
11	0.63	0.017	0.0018	0.00012	–	7338	2025	334	83	–	220	45.1
21	0.17	0.017	0.0018	0.00012	–	8647	823	295	67	–	168	48.3
31	0.30	0.023	0.0058	0.0018	0.00012	8300	1151	242	106	57	144	49.6
41	0.30	0.023	0.0058	0.0018	0.00012	8335	1143	240	101	55	126	49.4
51	0.30	0.023	0.0058	0.0018	0.00019	8361	1133	233	100	45	128	49.9
61	0.30	0.023	0.0058	0.0018	0.00019	8375	1125	228	96	47	129	49.4
71	0.30	0.023	0.0058	0.0018	0.00019	8383	1124	225	95	44	129	49.0
81	0.30	0.017	0.0018	0.00015	–	8393	1170	244	48	–	145	44.1
91	0.30	0.017	0.0018	0.00015	–	8397	1172	240	47	–	144	43.3
101	0.30	0.017	0.0018	0.00015	–	8412	1167	235	42	–	144	43.2
111	0.30	0.017	0.0018	0.00015	–	8411	1169	237	41	–	142	42.8
121	0.30	0.017	0.0018	0.00015	–	8417	1166	234	39	–	144	42.6
131	0.30	0.017	0.0018	0.00015	–	8426	1158	229	40	–	147	42.0
141	0.30	0.017	0.0018	0.00015	–	8426	1158	229	40	–	147	42.0
151	0.30	0.017	0.0018	0.00015	–	8427	1157	229	39	–	148	41.7
161	0.40	0.021	0.0018	0.00015	–	8205	1348	260	38	–	149	41.5
171	0.40	0.021	0.0018	0.00015	–	8207	1345	256	37	–	155	40.8
181	0.40	0.021	0.0018	0.00015	–	8207	1345	257	36	–	155	40.7
191	0.40	0.021	0.0018	0.00015	–	8212	1342	256	35	–	155	40.3
201	0.40	0.021	0.0018	0.00015	–	8212	1342	254	35	–	157	40.0
211	0.40	0.023	0.0018	0.00015	–	8213	1320	271	36	–	160	39.3

TABLE 6.5: Speed and coverage for the parameters produced by the coverage optimisation algorithm.

	Supertag Accuracy (%)		Ambiguity (cats / word)	F-score (%)	Speed (sent / sec)
Set	Single	Multiple			
			WSJ		
Baseline	91.14	96.07	1.27	83.41	48.5
Cutoff 51	91.14	94.13	1.10	83.15	49.9
			Wikipedia		
Baseline	89.8	95.4	1.3	82.5	46.31
Cutoff 51	89.8	93.2	1.1	81.3	53.6

TABLE 6.6: Performance of the best parameters produced by the coverage optimisation algorithm.

However, we are still faced with the problem of accuracy as opposed to coverage. A simple approach would be to shift the two endpoints defining our line segment to cover only the values where accuracy is above some threshold. While this idea is valid for the smaller beta value, it is not for the larger one. Moving the larger value is not possible because that does not prevent the sentence from being parsed in that region, and we may choose a beta level in that region as the result of some other sentence. Therefore we cannot guarantee that the level within the shortened line segment is the level that sentence will be parsed at. We can move the lower value because the beta levels will be considered in decreasing order and so the sentence will be parsed before we reach any beta levels in the ignored lower region.

The algorithm developed here may be a step in the right direction, but it is not a complete solution to the challenge of optimising these parameters.

6.9 Summary

Optimisation of the beta levels and tag dictionary cutoffs used by the supertagger leads to considerable improvements in efficiency. Developing a method of optimisation is made difficult by the fact that there are five levels and while most sentences exhibit similar behaviour at all levels, some will be parsed progressively less accurately, while others are parsed progressively more accurately. The investigation presented in this chapter is the first attempt to systematically explore this behaviour. If we can use these observations to improve the interaction between the parser and supertagger we will obtain significant speed and accuracy benefits.

The architectural developments described in the previous two chapters have enabled the creation of models with more features as we are no longer bound by the amount of RAM on a single machine. This chapter also explored a range of features, including feature sets that produced a statistically significant improvement in recall. In this way we are able to use adaptive training to improve accuracy as well as speed.

CHAPTER 7

Conclusion

7.1 Future Work

A wide range of directions exist for extension of this work. There is great scope for development of the algorithms, many other domains to apply the adaptation training method to, and the exploration of parameter optimisation here lays the groundwork for a great deal of further investigation.

The most direct extension would be to apply these adaptive training methods to the biomedical domain. A large collection of documents from the domain and an evaluation set already exist, making it a sensible next step. Cross-corpus tests on the WSJ, Wikipedia and the PubMed corpus would provide greater evidence for the adaptation results presented here.

The pre-processing methods of sentence boundary detection, tokenisation used here were very simple, which may be decreasing the quality of the automatically labelled data. The next stage in the process, using the parser to choose tags would also be worth investigating further. Since this stage only needs to run once to produce the training set it may be worth loosening restrictions on the parser to produce better results at a slower rate. This would provider greater scope for exploring which sentences provide the most benefit for adaptive training.

This work has clearly demonstrated that gold standard data can make a significant difference to accuracy on newspaper text, even when it makes up less than 1% of the training data. Open questions include how much gold standard data is needed to make a significant difference, and how much difference data with gold standard POS tags and automatically assigned supertags would make. Further, is adaptive training of the POS tagger possible? Specifically, using the supertagger to inform the POS tagger in the same way that in this work the parser has been used to inform the supertagger. In this case the aim would be to produce the set of POS tags that will lead the supertagger to produce a more accurate set of supertags.

Now that we have a range of different algorithms for model estimation it would be worth performing co-training experiments. Each model could be trained on the gold standard data, then used to parse extra data incrementally. The results of all of the models could be compared and when there is complete agreement, or at least a majority, the instance would be added to the training set and the process repeated. Another option would be to simply feed the results from one model into all others, effectively allowing all of the models to develop with some cross-pollination of tagging behaviour. A simpler method that takes a different tack would be to train a model using one algorithm, then use the final weights as the initial weighs for another algorithm, perhaps leading to a better local optimum. Finally, the whole training process could be kept entirely separate and the parser could determine the tag set by using multiple taggers, one for each model.

Additionally, the current system takes the weights produced by the perceptron algorithms, normalises them and treats them as a probability distribution in the same way as the weights from GIS and BFGS are treated. The results here have shown that while fast, single tagging is not accurate enough to be effective, but perhaps a multi-tagging perceptron algorithm could be implemented. For example, instead of producing tag–word pairs, the model could produce tagset–word pairs, and the predicted tagsets would be judged correct simply if they contained the correct tag. This particular method would dramatically increase the number of possible labels and so may be infeasible, but other approaches, such as always providing tags with weights within a certain ratio of the top tag and then adjusting both if neither is correct, may be effective.

The training method for the perceptron based algorithms is incremental, taking an existing model and adjusting it after each observation. This means they could continue to adapt as the parser is running, effectively learning continuously. It would be interesting to apply this method to the adaptive training experiments here and measure performance and changes in tagging behaviour over time, to observe the tagger adapting to the new domain.

These algorithms are also very flexible, and another interesting variation would be to tag an entire sentence at once by combining sums of local features to effectively create global features. Even at a local scale there is great scope for further investigation. While the initial tests here have not lead to overwhelmingly positive results, there is a great deal of extra data that could be used. The variety of features considered was also somewhat limited, partly due to the current architecture for defining features. It would be interesting to explore the use of 64-bit hashes as representations of features to enable the use of a more diverse range of features. For example, features that encode the presence of one

attribute and the absence of another. Also, many of the current features contribute little to the model, with weights close to or exactly zero. Feature selection could improve performance, or at least free up memory usage by the supertagger, potentially enabling the use of other, more exotic features.

The exploration in the analysis chapter laid the groundwork for a great deal of work not only for the supertagger, but also for the parser. The challenge of developing an effective and well founded method of optimising the beta levels and cutoffs remains unsolved. The exploration of some of the stranger cases indicated that there are specific issues in all areas, right from the POS tagger up. The length-based analysis suggested that different settings depending on length may be worthwhile. It would also be interesting to compare the behaviour of various models.

7.2 Contributions

I have demonstrated that adaptively training a supertagger boosts parsing speed and accuracy considerably. I have demonstrated that perceptron based algorithms can estimate model weights just as well as maximum entropy methods. I have taken the first steps towards a methodical exploration of the effects of beta levels on parser behaviour. And in the process of completing this work, I have made significant changes to a high performance state-of-the-art system, including implementing new weight estimation methods, and parallelising the supertagger training process.

The key contribution is the demonstration that adaptive training improves parser efficiency. The strategy of constructing a supertagging model that second-guesses the parser is novel and clearly effective. Importantly, these effects are not confined to a single domain, but have been demonstrated on Wikipedia text as well as newspaper text. It is also clear that the supertagger is adapting to the particular style of the domain it trains on, as models trained on Wikipedia perform poorly on the Wall Street Journal, and vice versa.

These experiments were not possible previously because of memory and time constraints. The second major contribution of this work was the implementation of alternative algorithms and a parallel form of the training process. These developments enabled the use of far larger data sets within the same training time and provided access to more RAM. As well as implementing these methods, this work has demonstrated that they produce models that are just as accurate as previous models.

The third significant contribution is an exploration of the supertagger and parser interaction. No such comprehensive analysis has been performed previously and the results of this work indicate that considerable gains in performance are possible if an effective means of optimising the parameters can be found. As part of this work an algorithm to optimise coverage was invented and the challenges of extending the method to optimise performance were explored.

Initially the system produced an F-score of 83.41% and ran at 48.5 sentences per second on the Wall Street Journal and produced an F-score of 82.5% and ran at 46.3 sentences per second on Wikipedia. The supertagging model that produced these results took two hours to train, using only forty thousand sentences. Using MIRA a model was trained in six hours but on one hundred times as much data, leading to an F-score of 83.99% and speed of 90.2 sentences per second on the WSJ. Another model trained on Wikipedia instead of newspaper text was able to parse 60.5 sentences a second and achieved an F-score of 83.3%. These models are just as accurate as the original system, but the first is 86% faster on newspaper text and the second is 30% faster on Wikipedia, clearly demonstrating that adaptive training leads to considerable performance improvements. These developments will lead directly to improvements in many other Natural Language Processing systems that rely on the output of state-of-the-art parsers, such as Question Answering systems.

Bibliography

Srinivas Bangalore. 1997. Performance evaluation of supertagging for partial parsing. In *Proceedings of the Fifth International Workshop on Parsing Technologies*. Boston.

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

L. E. Baum and T. Petrie. 1966. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

Steven Bird, Edward Loper, and Ewan Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

Avrim Blum and Tom Mitchell. 1998. Combining labeled and unlabeled data with co-training. In *COLT' 98: Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100. ACM, Madison, Wisconsin, United States.

Peter F. Brown, John Cocke, Stephen A. Della Pietra, Vincent J. Della Pietra, Fredrick Jelinek, John D. Lafferty, Robert L. Mercer, and Paul S. Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

C. G. Broyden. 1970. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6:76–90.

Raman Chandrasekar and Srinivas Bangalore. 1997a. Gleaning information from the web: Using syntax to filter out irrelevant information. In *World Wide Web, Stanford University*.

Raman Chandrasekar and Srinivas Bangalore. 1997b. Using supertags in document filtering: The effect of increased context on information retrieval effectiveness. In *In Proceedings of Recent Advances in NLP (RANLP) '97, Tzigov Chark*.

Raman Chandrasekar and Srinivas Bangalore. 1997c. Using syntactic information in document filtering: A comparative study of part-of-speech tagging and supertagging. Technical report.

Eugene Charniak. 1997. Statistical parsing with a context-free grammar and word statistics. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*. AAAI Press / MIT Press, Menlo Park.

John Chen, Srinivas Bangalore, Michael Collins, and Owen Rambow. 2002. Reranking an n-gram supertagger. In *Proceedings of the TAG+ Workshop*, pages 259—268. Venice, Italy.

John Chen, Srinivas Bangalore, and K. Vijay-Shanker. 1999. New models for improving supertag disambiguation. In *Proceedings of the 9th Meeting of EACL*, pages 188–195. Bergen, Norway.

N Chinchor. 1992. Statistical significance of muc-6 results.

Stephen Clark. 2002. Supertagging for combinatory categorial grammar. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, pages 19—-24. Venice, Italy.

Stephen Clark, Ann Copestake, James R. Curran, Yue Zhang, Aurelie Herbelot, James Haggerty, Byung-Gyu Ahn, Curt Van Wyk, Jessika Roesner, Jonathan K. Kummerfeld, and Tim Dawborn. 2009. Large-scale syntactic processing: Parsing the web. Technical report, JHU CLSP Workshop.

Stephen Clark and James Curran. 2007a. Perceptron training for a wide-coverage lexicalized-grammar parser. In *ACL 2007 Workshop on Deep Linguistic Processing*, pages 9–16. Association for Computational Linguistics, Prague, Czech Republic.

Stephen Clark and James R. Curran. 2003. Log-linear models for wide-coverage ccg parsing. In *Proceedings of the 2003 conference on Empirical methods in natural language processing*, pages 97–104. Association for Computational Linguistics.

Stephen Clark and James R. Curran. 2004. The importance of supertagging for wide-coverage ccg parsing. In *Proceedings of 20th International Conference on Computational Linguistics (COLING)*, pages 282–288. Geneva, Switzerland.

Stephen Clark and James R. Curran. 2007b. Wide-coverage eficient statistical parsing with ccg and log-linear models. *Computational Linguistics*, 33(4):493–552.

M. Collins and Y. Singer. 1999. Unsupervised models for named entity classification. In *Joint SIGDAT Conference on Empirical Methods in NLP and Very Large Corpora*.

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

Michael Collins and Brian Roark. 2004. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Meeting of the ACL*, pages 111—118. Barcelona, Spain.

Nicholas Cooper. 2007. *Improved Statistical Models for Supertagging*. Ph.D. thesis.

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

James R. Curran. 2003. Blueprint for a high performance nlp infrastructure. In *SEALTS '03: Proceedings of the HLT-NAACL 2003 workshop on Software engineering and architecture of language technology systems*, pages 39–44. Association for Computational Linguistics, Edmonton, Canada.

James R. Curran and Stephen Clark. 2003. Investigating gis and smoothing for maximum entropy taggers. In *Proceedings of the 10th Meeting of the EACL*, pages 91—98. Budapest, Hungary.

J. N. Darroch and D. Ratcliff. 1972. Generalized iterative scaling for log-linear models. *The Annals of Mathematical Statistics*, 43(5):1470–1480.

R. Fletcher. 1970. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322,.

Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and Systems Science*, 55(1):119–139.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

N. Ge, J. Hale, and E. Charniak. 1998. A statistical approach to anaphora resolution. In *Proceedings of the Sixth Workshop on Very Large Corpora*, pages 161—-170. Montreal, Canada.

D. Goldfarb. 1970. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24:23–26.

Sally A. Goldman and Yan Zhou. 2000. Enhancing supervised learning with unlabeled data. In *ICML '00: Proceedings of the Seventeenth International Conference on Machine Learning*, pages 327–334. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. 1996. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, 22(6):789–828.

Julia Hockenmaier. 2003. *Data and Models for Statistical Parsing with Combinatory Categorial Grammar*. Ph.D. thesis, University of Edinburgh.

Julia Hockenmaier and Mark Steedman. 2001. Generative models for statistical parsing with combinatory categorial grammar. In *ACL '02: Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, pages 335–342. Association for Computational Linguistics.

Baden Hughes, Srikumar Venugopal, and Rajkumar Buyya. 2004. Grid-based indexing of a newswire corpus. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 320–327. IEEE Computer Society, Washington, DC, USA.

E. T. Jaynes. 1957. Information theory and statistical mechanics. *Physical Review*, 106(4):620–630.

Aravind K. Joshi and Srinivas Bangalore. 1994. Disambiguation of super parts of speech (or supertags): almost parsing. In *Proceedings of the 15th conference on Computational linguistics*, pages 154–160. Kyoto, Japan.

Jun'ichi Kazama and Kentaro Torisawa. 2008. Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations. In *Proceedings of ACL-08: HLT*, pages 407–415. Association for Computational Linguistics, Columbus, Ohio.

Raymond Lau, Ronald Rosenfeld, and Salim Roukos. 1993. Adaptive language modeling using the maximum entropy principle. In *HLT '93: Proceedings of the workshop on Human Language Technology*, pages 108–113. Association for Computational Linguistics, Princeton, New Jersey.

Wendy Grace Lehnert. 1977. *The process of question answering*. Ph.D. thesis, New Haven, CT, USA.

Robert Malouf. 2002. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Workshop on Natural Language Learning*, pages 49—55. Taipei, Taiwan.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993a. Building a large annotated corpus of english: the penn treebank. *Computational Linguistics*, 19(2):313–330.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993b. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

John D. Laffertyand Andrew McCallum and Fernando C. N. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML '01: Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective self-training for parsing. In *Proceedings of The Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

P. Miller. 2002. pympi–an introduction to parallel python using mpi. *Livermore National Laboratories*, 11.

R. Mitkov. 2002. *Anaphora Resolution*. Pearson Education.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In Eric Brill and Kenneth Church, editors, *Proceedings of the Empirical Methods in Natural Language Processing*, pages 133–142. Philadelphia, Pa. USA.

F. Rosenblatt. 1958. The perceptron - a probabilistic model for information - storage and organization in the brain. *Psychological Review*, 65(6):386–408.

Kenji Sagae and Alon Lavie. 2005. A classifier-based parser with linear run-time complexity. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 125–132. Association for Computational Linguistics, Vancouver, British Columbia.

Anoop Sarkar. 2001. Applying co-training methods to statistical parsing. In *NAACL '01: Second meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies 2001*, pages 1–8. Association for Computational Linguistics, Pittsburgh, Pennsylvania.

Anoop Sarkar. 2007. *Combining Supertagging and Lexicalized Tree-Adjoining Grammar Parsing*, chapter TBC, page TBC. MIT Press.

Anoop Sarkar, Fei Xia, and Aravind Joshi. 2000. Some experiments on indicators of parsing complexity for lexicalized grammars. In *Proceedings of COLING*, pages 37–42.

D. F. Shanno. 1970. Conditioning of quasi-newton methods for function minimization. *Mathematics of Computation*, 24:647–656.

M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra. 1995. *MPI: The Complete Reference*. MIT Press, Cambridge, MA.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.

A. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269.

David Yarowsky. 1995. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd annual meeting on Association for Computational Linguistics*, pages 189–196. Association for Computational Linguistics.