Regular Treebank Generalisation

Isaac Sijaranamual

August 19, 2007

Contents

Preface

In fulfilment of my doctorandus.

I will *not* perpetuate the use of 'we' in, what is supposed to be, an original piece of work by a single person. And no, it does not 'make it look more scientific.'

Introduction

A longstanding goal since the inception of computers, is for the machine to be able to comprehend language in a way we humans do. What seems like a simple thing for most of us turns out to be a hard problem to solve. So hard in fact that it still has not been solved. This fact is not that surprising given the fact that it is still poorly understood how humans process language. Indeed, language is perceived to be a direct reflection of the mind that produces it. As capable as we are when it comes to using language, so incapable are we at expressing how we do it. It is something that happens automatically. We only become aware of this fact once we try to explain it to someone else, or to a computer.

If we are to instruct these machines how to deal with or to process language then it would be a good start to understand what it is we are trying to explain. As is common when scientists try to understand a phenomenon, they try to take it apart and try to study the parts in isolation. Linguistics, the study of language, is no different. And as with any research field it is wide and varied, but too broad for a single person, to treat within the confines of a thesis. Therefore I will restrict my attention to the description and understanding of grammar. Or more precise, the syntax of natural language.

As noted as early as the beginning of last century, a peculiar property of the aspect of language, the words, manifests itself. If you count the occurrences of all words, and plot the logarithm of the frequency against the logarithm of the rank of each word then there is an approximate linear relationship between the two. This is know as the Zipf or power law.* He discovered that in normal language there are a few very common words and a whole lot of infrequent ones,unique word occurrences typically make up at least half of the all different word types. What this means is that one is bound to encounter new words in any significant piece of new, non-trivial, text. No matter how much text one has seen.

An even stranger fact is that this does not only happen at word level, but also at the level of syntactic structure. The types of sentence structures one encounters

*named after Zipf for his treatise in Zipf [1932], but earlier references go back as far as Estoup [1916] also exhibit this long tailed distribution that governs the distribution of the words. This fact was noted by Sampson [1987].

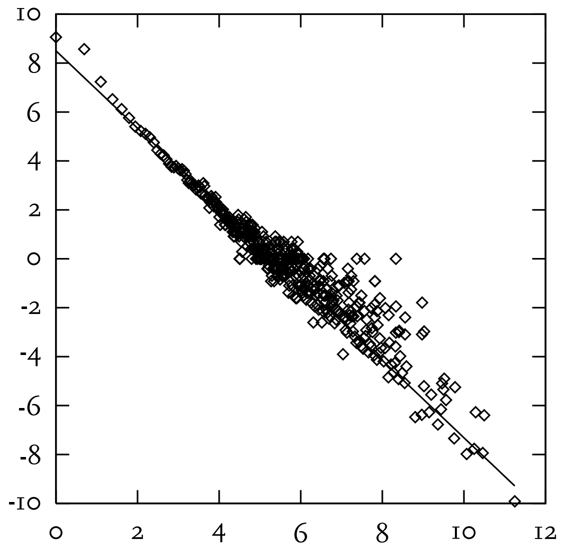


Figure 1: Log-log plot of smoothed count of counts of PCFG rules extracted from WSJ sections 02–21. Of the 14981 grammar rule types, 8610 (that is approximately 57%) occur only once.

How can we make sense of these boundless collections of things? For the words the answer usually boils down to lexical similarity (word form) or context (syntactic/semantic role). We tend to assign meaning, however vague, to new unknown words based on similarity to those that we do know. This similarity can manifest itself in the literal similarity or the context in which it appears. At least we assume that they share a meaning because they are somehow 'similar'. But what about the syntactic rules? Do they also have some notion of similarity? I will argue that they do.

The similarity of the rules will be based solely on their internal structure. I will treat these rules as simple sequences of symbols. A notion of similarity of rules can then be based on that of similarity between lists or sequences of symbols.

A simple measure of similarity between sequences is the n-gram or Markov models. The n-gram models are models that count all subsequences of n consecutive symbols in a sequence. These counts can be used to directly define similarity between two sequences, simply count how often n-grams from one occur in the other. Or indirectly, in which case the n-grams are used to define (conditional) probabilities with respect to a sequence. This probability can then assign probabilities to other sequences and in effect measure the similarity between them.

Assume that we have extracted set of rules from a sample of language. If we take another sample of that same language, we expect to see a lot of rules we had already seen in our first sample, but also a significant set of new rules. After all it is extremely unlikely that our first sample was complete with respect to that language. This means that analysing the new sample with the old rules would give a markedly different result than analysing it anew. It turns out this keeps happening each time you take a new sample of the language. Assuming that there is indeed structure in the language we can capture, we seem to be missing it. It would be nice if we could infer the right structure from the sample(s) we do have.

This brings us back to the similarity between rules, for most of the rules of the old sample will occur in the new sample, and the new rules that we had not seen before usually resemble one or more old ones.

Because of their simplicity and good performance, n-grams are ubiquitous in computational linguistics. The idea is so simple that it is too tempting to try to improve on.

This thesis will look at the extension of the n-grams, the finite state machines, to see how much we can gain by using a more powerful model to describe sequences.

Very well, thesis statement(s):

1. How can we sensibly generalise treebank grammars?
2. Can we escape the clutches of the omnipresent n-grams?

Overview

The thesis is divided as follows. In chapter two I will describe previous work, outlining the various parsers and identify their shortcomings. The next chapter deals with grammar inference. Chapter four treats in detail the models I devised, with the results in the following chapter. And the final chapters wraps up the discussion and provides pointers for future research.

1 Background

This chapter provides an overview of the field of computational linguistics and where this thesis fits in.

1.1 Computational Linguistics

The aim of computational linguistics is to find models of natural language, specified in enough detail such that it can be run by a computer. The models can be based on one's own intuition or on that of a linguist.[*] Once such a model has been formulated it should be critically inspected, implemented and suitably tested.

As with any scientific endeavour, if the results of the test indicate that the original model is significantly deficient in any way (shape or form), it can be refined and tested until it satisfactorily explains the language of interest.

The syntactic analysis of a sentence structure is known as parsing. Parsing sentences, or more generally any text, requires that we have a systematic way of describing that structure. The grammar of a language describes how the separate parts, usually words, fit together to form the constructs that we in our daily lives recognise as language.

This seems straightforward enough at first sight, but the apparent simplicity is deceptive. Notwithstanding the long history of people trying to come to a comprehensive description of language, any language, all have failed to varying degrees.

For example, most of the structure of the English language is described by a succinct context free grammar, but there are always examples, utterances that elude any such given grammar. This has driven linguists both away from and beyond these grammars. It is still unknown whether natural language even has a 'proper' grammar. In any case it has proven useful to have, and use, even crude grammars for any given language.

Current day natural language processing has seen the (successful) convergence of both the formal grammar and statistical methods. The basic idea is to take the

[*]The present author does not consider himself to be one.

crude, sloppy grammars, and refine them by appropriately assigning probabilities to its rules. These probabilities can encode a subjective *a priori* notion of confidence in a given part of the grammar, or it can be used to describe a usage pattern in a sample of the language.

There are two obvious ways of obtaining a grammar, one is by explicit specification and the other by 'manual parsing'. The former method generally produces small, simple and understandable grammars. But these grammars have a tendency to be too restrictive, that is, they cannot assign a consistent structure to arbitrary text. The grammars produced by the latter method tend to be of an ad hoc nature, causing them to be larger and messier. The upside is that they are better capable of handling new, unseen, text.

A major part of the differences between the two is due to their redundancy. Grammarians usually try their best to create frugal grammar(s). Ideally everything expressible by the grammar can be expressed in one way and one way only. Clearly, small and simple systems are more easily analysed and reasoned about. Whereas the implicitly defined grammars produced by a manual parse of a text will have a multitude of ways to analyse the same sentence.

Different analyses of the same sentence might not be desirable, and even if they are, then they might not be equally likely. Sentences with different meanings have different internal structure. The converse is mostly true: sentences with a different internal structure tend to have a different meaning. The problem is that this can happen even when two sentences are identical at the word level, see figure 1.1. This gives rise to (one aspect of) the well known ambiguity of natural language.

By using the statistics gathered for the language, we can put a number, a probability, on the different analyses. In essence it allows us to disambiguate via structural preference. It is unlikely that simply plugging the numbers will provide the acuity to correctly resolve all these ambiguities, but it does provide a powerful tool.

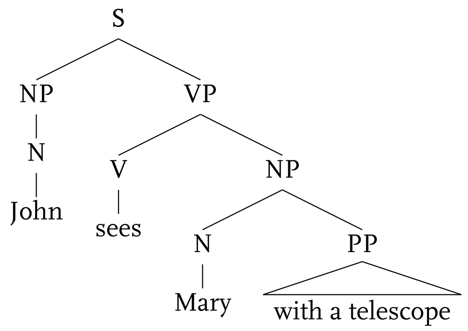
Statistics can only be calculated for a suitable collection of text. These collections, or corpora, are a selective subset of the language one would want to study and are the subject of the next section.

1.2 Corpora and Annotation

Once a language sample has been collected, it can be analysed and annotated. I shall restrict this discussion to the syntactic annotation scheme used in the Penn Treebank (PTB) corpus. Or specifically a restricted subset of that annotation scheme.

At the lowest level we have the so-called parts-of-speech, or POStags. A simplified tag set can be seen in table 1.1. The actual set of tags used in the PTB has a

$$\begin{array}{lcl} \text{S} & \rightarrow & \text{NP VP} \\ \text{NP} & \rightarrow & \text{N} \\ \text{NP} & \rightarrow & \text{N PP} \\ \text{VP} & \rightarrow & \text{V NP} \\ \text{PP} & \rightarrow & \ldots \end{array}$$



$$\begin{array}{lcl} \text{S} & \rightarrow & \text{NP VP} \\ \text{NP} & \rightarrow & \text{N} \\ \text{VP} & \rightarrow & \text{V NP PP} \\ \text{PP} & \rightarrow & \ldots \end{array}$$

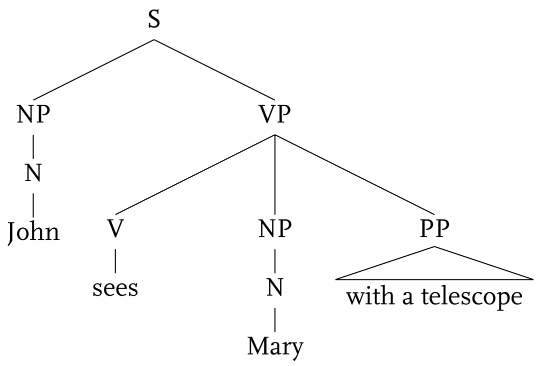


Figure 1.1: Syntactic ambiguity for "John sees Mary with a telescope". Two different parsetrees, both with different meanings

total of 72 tags; 45 POStags and 27 syntactic categories.[†] If the specificity of the tags is too coarse, we might miss valuable information, too fine and it becomes unwieldy.

Annotating a text with parts of speech is a reasonably simple job.[‡]

The next step up in terms of (linguistic) expressiveness is annotating the text with chunks, grouping consecutive words together in single units. This chunking process is also known as shallow parsing. The typical process is to identify constituent phrases in a sentence, like noun- and prepositional phrases, and marking them as such. This introduces a new set of tags, syntactic tags or syntags. The big benefit of chunking is that it is still a relatively simple process. No nesting or overlapping of chunks is allowed, this is what sets it apart from the next step.

[†] All versions of the different documents supplied with the PTBv2 CD give different counts. The numbers reported are the ones in the dataset that I used.

[‡] Simple in the meaning that once the set of POS tags is fixed, per-annotator and inter annotator disagreement is very low. Furthermore it is very easy to write a computer program to get a tagging score upwards of 90% precision (accuracy?).

Table 1.1: A simple set of parts-of-speech.

VB	verb
N	noun
DT	determiner
JJ	adjective
RB	adverb
CC	conjunction
UH	interjection

In full parsing, or structural analysis of sentences, 'chunks' can be arbitrarily nested.[§] The number and type of tags to be used in the structural analysis is still unknown. It may range from a handful to well into the hundreds (or even thousands).

Table 1.2: English corpora

name	created	period	size (Mwords)	description
Brown	1967	1961	1	American English, created by Kucera and Francis; described in Computational Analysis of Present-Day American English (1967)
BNC	1991–1994	late 20th	100	designed to represent a wide cross-section of British English from the later part of the 20th century, both spoken and written
PTB	1989–1992	1991	5	full PTB includes the Brown, see 1.3

The prime object of interest for this thesis will be the Penn Treebank (PTB). Or more specifically, the Wall Street Journal (WSJ) subsection. A small overview of the Penn Treebank is shown in table 1.3.

The reason that I use the WSJ is simple, it is the *de facto* standard in the parsing community. It was one of the first large, syntactically annotated corpora available.

[§] but they may not overlap one another, i.e., one constituent may either completely contain another or be completely disjoint with it

Table 1.3: Penn Treebank in detail, taken from [Marcus et al., 1994]

Description	annotation	
	POS (Tokens)	parsed (Tokens)
Dept. of Energy abstracts	231,404	231,404
Dow Jones Newswire stories	3,065,776	1,061,166
Dept. of Agriculture bulletins	78,555	78,555
Library of America texts	105,652	105,652
MUC-3 messages	111,828	111,828
IBM Manual sentences	89,121	89,121
WBUR radio transcripts	11,589	11,589
ATIS sentences	19,832	19,832
Brown Corpus, retagged	1,172,041	1,172,041
Total:	4,885,798	2,881,188

Having a standard corpus to train, test and validate on has been a great boon to the parsing community. Since the dataset used by everyone is identical, results can be compared directly and any difference in the results is a direct consequence of the particular method used and not due to some difference in the data. But this also creates its own set of problems, as the parsers start getting better, the improvements start getting smaller and smaller, to the point where one has to ask the question to what extent these numbers are still meaningful. I want to point out that the start of this chapter stated that we were interested in constructing models of language (English) and not models of mid-nineties Wall Street Journal articles. Alas, this is a problem, but one well outside the scope of this thesis.

1.3 Parsing

The basis for the models in this thesis is a simple PCFG which is extracted from the treebank. As described in Charniak [1996] this model performed significantly better than was commonly believed at the time. This belief was based on the experience of the poor performance of handcrafted grammars. Couple this to limited computing power and the immense size of the treebank grammar, had led many to disregard the possible utility of treebank grammars.

Around the same time Collins published the results obtained with his dependency parser. In [Collins, 1996] he describes his parser, which is based on the bilexical dependencies between head and modifier words. The dependencies are further specialised by defining a ternary relation over the phrasal categories (syntags)of the constituents of both words and the category that the dependency in question is part of (this also allows him to reconstruct the original parsetree). As a final addition he adds the 'distance' between the lexical items. The author describes the necessity of smoothing to overcome the data sparsity that conditioning on such large and specific context entails.

Better results still, but at the cost of a more complex parsing model, have been reported by Collins [1999]. In his dissertation he expands his earlier model from 1996 by adding (among other things) a regular PCFG grammar, a zeroth order Markov model to generate the rules of that grammar,[¶] subcategorisation frames, and most importantly, more sophisticated smoothing to deal with the significantly increased context on which he conditions.

Earlier work by Magerman, his SPATTER parser [Magerman, 1995], works by constructing a decision tree to classify each word and its relation to other words.

The fact that both the SPATTER parser from Magerman and Collins' parser were lexicalised and outperformed the simple, unlexicalised, treebank grammar led researchers to believe that lexicalisation (both head lexicalisation and the bilexical dependencies) is of vital importance to parsing. But more detailed analyses of the effects of various enhancements of parsing models, among which the head- and dependency lexicalisation, by various researchers [Bikel, 2004, Gildea, 2001] show that their effects are small and in the case of bilexical dependencies even corpus specific.

The main reason that Collins's model is able to perform so well is because it is capable to make very fine distinctions over the similar-but-different instances of syntactic tags, through the system of conditioning on both the heads and the subcategorisation frames. This enables him to differentiate the subtle nuances of the differences in usage even within the same wordclass. Therefore considerable effort is put into smoothing, primarily via back-off to coarser models.

A different approach has been the work of Johnson [1998], who has also put considerable effort in investigating the relationship between treebank annotation(s) and PCFG parsers. He examines various tree transforms: increasing the depth of the flat noun- and verb phrases (making the grammar more regular, but increasing the independence), flattening recursive noun- and verb phrases (the opposite effect) as well as parent annotation (adding the tag of the parent of each node to the node itself thereby increasing specificity of the symbols).

This has been used to great effect by Klein and Manning [2003], who reach levels of performance with their unlexicalised parser comparable to the best early lexicalised parsers. The basics of which is Markovising not only over grammar rules, but also over node ancestry, where Johnson only considered the parent node, they also condition on the grandparents. In addition to that they also did semi-

[¶]which is odd, since English has a relatively strict word order, exactly something that low (zeroth) order Markov models are not capable of capturing.

automatic symbol splitting, again creating more specialised symbols.

Petrov et al. [2006] take this a step further and manage to have their unlexicalised parser outperform all other published parsers.

A different approach has been taken by Eisner in his PhD thesis, [Eisner, 2001]. His transformation model extracts flat PCFG rules, with he head of the rule replaced by the lexical item it represents. He then constructs a graph where each rule is connected to its edit distance-1 neighbours. A weight is associated with each type of edge, where the types are: insertion, deletion and substitution[||] and thus with each of these edges.

1.4 Parse Evaluation

In parsing text the aim is to give a reasonable analysis of the input text most of the time. Once that goal has been reached, the requirements quickly shift to always giving reasonable analysis, but also a correct one most of the time. The notion of reasonable and correct depends on the domain in which the system is used and the ultimate goal one has envisioned. For instance, for the task of information retrieval, one is mostly interested in discerning nouns and verbs, or the slightly more complex task of named entity recognition to be able to return a set of relevant documents. Similarly simple language models can be employed by speech recognition engines. For more taxing problems like question-answering (related to information retrieval, but instead of returning a set of relevant documents, Q&A systems try to answer the query phrase the user entered) or machine translation a more detailed analysis is needed to be of any help.

While evident, this does pose a big problem, namely, that of evaluation. How do we compare one parse against the other? Before the widespread use of treebank parsing (and quite a while after their they had shown their use/value), the only grammars were handcrafted. Linguists sat down and wrote up these complex rule sets, describing to the best of their abilities the structure of their language. This puts the burden of deciding what is and is not correct solely on the creator(s) of the grammar. This is a reasonable and clear-cut solution, so why abandon this approach?

Although the handcrafted grammars are considered to be of high quality, creating them is time-consuming and the result is very fragile. They tend to break down when applied to realistic, i.e., real world data. This is either due to unforeseen combinations of sentence constructions or even 'incorrect' language usage.[**]

[||]but also any of the more specific transformations: insertion/deletion/substitution of syntags by class and i/d/s of lexical items.

[**]incorrect with respect to some idealised grammar, which is usually *also* specified by linguists.

Since most people prefer a partially correct answer to no answer at all, people have looked to other solutions, one that still uses the familiar notion of grammar, but which does not have the drawbacks of the hand built ones. From the empiricist's point of view correctness of natural language is determined by its usage. There is only a small set of basic rules for each language, the details of which shift over time.

Corpus based approaches to natural language processing fit neatly into this empirical framework. The annotators of the corpus mark up sample sentences based on an implicit grammar. Even though producing a consistent annotation over the entire group of annotators is a hard problem (the PTB annotation manual is a couple of hundred pages), it turns out to be easier than creating handcrafted grammars.

With this treebank at hand, parsing now becomes an instance of supervised machine learning. The machine needs to learn what we humans consider to be the structure of language. And it does so by being provided with a large set of examples of 'correct answers' (desirable would probably be a better word.)

But this reintroduces the problem of evaluating parse performance. The most obvious solution, and the one used in supervised machine learning, is to declare the example set as the one true answer, the 'gold standard'. Parsers can then be rated on their ability to match whatever examples it is given. This is the intrinsic performance of parsers. If the score attained by the parser is 100% then there is no question, it is doing at least as good as the human who annotated the data would do. But what if the parser is unable to deliver a perfect score each and every time?

This is where the extrinsic comes in. In the extrinsic evaluation of parsers one is not interested of the score of the parser in isolation, but as part of a larger system, for instance a document analysis system. In that case alterations to the parsing subsystem are evaluated by scoring the document analysis system as a whole. This, inevitably, gives better results because improvements in whatever score is used, directly benefit the entire system and is the measure one is usually interested in (unless you are a computational linguist, creating and studying parsers for the sake of it). The downside of extrinsic evaluation is that it can be too hard (too many variables) or just computationally intensive.

To this end, researchers have proposed various evaluation scores to do intrinsic performance evaluation. By far the most pervasive when it comes to parse evaluation is PARSEVAL, described in Black et al. [1991]. This metric is actually a set of related metrics, all of which depend directly on the tree structure or brackets. The most interesting quantities it encompasses are:

Labelled Recall the number of (labelled) parentheses in the gold standard that are matched by the proposed parse, divided by the total number of parentheses in the gold standard.

Labelled Precision the number of (labelled) parentheses in the proposed parse that match the parentheses in the gold standard divided by the total number of parentheses in the proposed parse.

Crossing Brackets The number of constituent boundary violations.

Furthermore, a common derived quantity is the FMeasure, which is the harmonic (unweighted) mean of the recall and the precision.

The aim for the PARSEVAL metric was to provide a single unified set of measures to evaluate a wide variety of grammatical formalisms, be it phrase structures or dependency grammars. It can even be applied to both stochastic and symbolic parsers.

2 Problem Statement

In this chapter will explicate the deficiencies of the standard treebank grammar and propose a solution.

2.1 Previous Work

In his paper on treebank grammars [Charniak, 1996], Charniak proposes five main reasons for the suboptimal performance of the raw treebank PCFG. They are, not having the correct rules, having the right rules but not the right probabilities, insufficient information in the POStag sequence (no lexicalisation), insufficient time/memory to consider all parses and finally an incorrect gold standard.

To test the effects of points one and two he runs the same test, but with the same set for both testing and training. The results of those tests show that having the correct rules and with the correct probabilities have a minor effect on the recall, but a much bigger effect on the precision. Labelled recall goes from 80.4% to 81.8%, a 7.1% error reduction and labelled precision from 78.8% to 83.3%, a 21.2% error reduction. He calls it 'an interesting question' why the results for the two measures differ that much. But he also claims that the big gains will most likely not come from making (incremental) improvements to the basic method, instead a productive (find another word) avenue of research would be the incorporation of lexical information. A fact supported by the significantly better results obtained by the lexical decision tree parser SPATTER, [Magerman, 1995], which attains 85LP/84LR.

Treebank woes

As we have already seen in the previous chapter, the treebank grammar performs reasonably well as-is (based on the work by Charniak [1996]). But in the same paper a couple of limitations were highlighted. The limitations can be attributed to a couple of factors.

The most obvious one being the inadequacy of PCFG models in general to capture the complex structural interdependencies of natural language. Some of the more powerful formalisms that have been proposed, and used, to address this problem are DOP [Scha, 1990], TAG, CCG, LFG. Although interesting lines of research, the scope of this thesis will remain focused on the PCFG formalism.

The other factor is the heavy influence of the details of the annotation style used to construct the treebank. In the Penn Treebank, the annotators chose to prefer a flatter style of tree annotation. The choice was heavily influenced by the ability of the annotators to chose between different representations. The non-choice of not using deeper, more nested, structure was a pragmatic neccesity with the limited (human) resources. The flatness of the rules has both beneficial and detrimental effects on the performance of simple PCFG parsers.

The beneficial effect is that by flattening the structure you remove structure which would have been missed by the PCFG due to its context-free independence assumption. The detrimental side effect of this is that it introduces a lot of new rules for all the flattened structure(s). Whereas in the non-flattened case the PCFG parser might have (over-)generalised, it now fails to identify similar –but non-identical– structures.

To overcome these deficiencies, and to thus attain better performance, it is necessary to change the underlying model. This may mean dispensing with the PCFGs all together, or by increasing its sensitivity to structural information. Another option would be to change (improve) the annotation you start with. This might seem like a stop-gap measure, or even 'cheating', one does after all change the problem definition, but this transformation step is usually performed transparently. What I mean by that is that the input and outputs to the parser remain the same, and hence it is evaluated on a 'compatible' domain. This can be understood to be a conversion layer, which sits between the parser and its surroundings. In this way we allow the parser to work with a much richer annotation, which it can then use to make better informed decisions. Finally the output of the parser (with the rich notation) is mapper back to the original form.

The final problem, and main subject of this thesis, has already been hinted at, but will now be made explicit. Since the treebank is a sample of English, it is subject to the same problems that govern all samples, statistical noise. It is clear that a poor approximation to the distribution of English (structures) will result in bad, or even outright wrong analyses of English sentences. A fact made worse by the abovementioned relatively flat rule annotation.

I consider the set of rules to be a representative sample of a simple language. I will therefore tackle the problem by using stochastic language inference. This is a well studied field so I do not expect any insurmountable problems to crop up.

In what follows I treat two methods. The first are the Markov models, what they are, how they are used and what their shortcomings are. The second method is that of regular language inference, what it does and how it solves the problems that the Markov models have.

Markov Models

The first method, the Markov models, are not so much a method of language inference as they are a method to describe the statistical properties of sequences of symbols. These sequences can be the letters of words, the words of a sentence, or as in our case, sequences of abstract symbols.

Since we are dealing with English, and due to the structure of English, once we know that we are currently looking at, say, a noun phrase, this greatly cuts down the possibilities of which tags can occur. Not only *which* tags can occur, but due to the strict word order of English, also in what order. A non-recursive noun phrase is likely to start with a determiner, and knowing that the previous symbol was a determiner increases the probability of a noun and makes any kind of punctuation highly unlikely. This is exactly what a first order Markov model does.

These 'rules' differ per constituent, but the important fact is that such regularity is present regardless. And as such it can, and should, be exploited. Now, to acquire these rules, all one has to do is to extract and count them. Once these statistics have been collected, these can be used to assign probabilities to rules in a manner similar to that of assigning probabilities to rules directly (by relative frequency). So instead of calculating the probability of rule $L \rightarrow R$ as

$$p_{RF}(L \rightarrow R) = c(L \rightarrow R) / \sum_i c(L \rightarrow R_i)$$

instead use

$$p_{\text{bigram}}(L \rightarrow R) = \prod_{r_i \in R} p(r_i | r_{i-1})$$

for all (pairs of) symbols in the rhs R of the rule. A full treatment and accompanying notation, is given in section 3.1.

An example. If we take a small subset, on the order of one tenth, of the corpus and use that to calculate the rule frequencies for the entire corpus, we get the following numbers. For the relative frequency- and the bigram estimate for the rule which rewrites a noun phrase as a determiner and a noun, NP $\rightarrow$ DT NN:

$$\begin{aligned} p_{RF}(\text{NP} \rightarrow \text{DT NN}) &= c(\text{NP} \rightarrow \text{DT NN}) / \sum_i c(\text{NP} \rightarrow \text{rhs}_i)) \\ &= 2944/31369 \\ &\approx 9.385 \cdot 10^{-2} \end{aligned}$$

and the bigram probability,

$$\begin{aligned} p_{\text{bigram}}(\text{NP} \rightarrow \text{DT NN}) &= \frac{c(\text{DT}|\text{start})}{c(\text{start})} \cdot \frac{c(\text{NN}|\text{DT})}{c(\text{DT})} \cdot \frac{c(\text{stop}|\text{NN})}{c(\text{NN})} \\ &\approx 6.766 \cdot 10^{-2} \end{aligned}$$

where the counts are only the counts collected from the noun phrases. We can see that the probability of the rule has been reduced significantly (as it happens this rule is actually the second most common one for noun phrases, only surpassed by the recursive NP $\rightarrow$ NP PP). This immediately highlights the drawback of using a first order Markov model, it is way too permissive. In accepting new rules, it takes away too much of the probability of the events it has seen (for the entire corpus the relative frequency estimate of the rule NP $\rightarrow$ DT NN is $9.371 \cdot 10^{-2}$). Increasing the model by a single order produces a significantly better estimate, but this introduces new problems, both a consequence of the increase in the number of free parameters of this larger model. Parameters which have tob e estimated from the data. The more data you have, the better the estimate. But if you cannot (easily) increase the amount of data you will have to make do with a simpler model, otherwise the estimates will be inaccurate with respect to the real probability distribution.

The usual solution to this problem is to construct all Markov models up to a fixed order, say three, and combining the probabilities of all these models. Combining these models can be as simple as using a fixed factor per order, or more complex, adjusting the factor relative to the evidence one has (even within a single order not all contexts occur with equal probability). These complex interpolation schemes produce very accurate estimates for the probabilities, but still have the large number of parameters.

Since I will be using the model as a generative one, the large number of parameters of a smoothed model would become a problem. The models will therefore not be smoothed (in the hoope that it will not be detrimental to the ultimate goal of parsing.)

Breaking the Rules

It should be clear that breaking up the flat rules of the treebank grammar is of considerable importance for both the estimation of rule probabilities as well as the coverage of the grammar. As has already been noted, increasing the tag set (making each more specific) and/or flattening the rules increases the need to do smoothing.

For the first problem any kind of (statistical) estimation or smoothing method could in principle be used. But the downside is that that leaves the second problem untouched.

The idea then is, is it possible to generate these, new, unseen grammar rules whilst smoothing out the occurrences of the ones we have seen?

The often used Markov models are one way of accomplishing exactly this. Not only do they smooth out the probabilities, but they also allow for new rules to be generated. The new rules are also types which resemble the ones we have already encountered before. So they solve both problems, albeit in a crude way, but they are simple, both conceptually as well as computationally. Training or estimating them from data is almost trivial, it is a simple matter of counting. By assigning different probabilities to events in a sequence based on the context of each symbol they allow the bigger joint probabilities to be broken up.

Since Markov models have a fixed context size, which is to be determined beforehand, this puts a burden on the implementer to decide which context size to use. Too small a value and you (might) miss important correlations between symbols further apart. Too large and you run into data sparsity problems, putting us right back at square one.

The usual solution to this problem is to smooth over various orders of Markov models of different order. Indeed, most smoothing methods used in natural language processing find their origin in ideas thought up to improve the basic Markov models. This is not, however, the most efficient solution. Since the number of parameters that must be estimated increases exponentially with the order of the model.

Therefore I propose to use the more expressive family of regular languages to overcome these problems. The regular languages are a proper superset of the Markov models and while retaining their efficiency.

Stochastic Regular Language Inference

Various methods have been devised to do regular language inference, or identification. They all follow the general procedure of acquiring a sample, assuming a specific model class (in this is case regular grammars) and finally trying to identify that model as best as possible, if it is even possible at all. The multitude of methods devised for the identification of regular languages can be attributed to two points, they are a simple formalism and as such can be efficiently implemented. And, equally important, the negative result of Gold [1967]. Which shows that it is impossible to learn regular grammars from positive data alone.

Since the main goal of this thesis is to be able to out do Markov models, it would be nice if the inference algorithm is at least capable of capturing the same structure as the Markov models. This rules out acyclic models, such as the k-reversible languages of Angluin [1982].

The SFSM inference algorithm that i have used is a state merging algorithm. In particular the MDI algorithm devised by Thollard et al. [2000], which itself is based on the ALERGIA algorithm of Carrasco Jiménez and Oncina Carratalá [1994, 1999]. Both algorithms work according to the same basic principle: 1) build a PTA from the sample set, 2) for all pairs of states, test their compatibility (stochastic equivalence) and, 3) merge if compatible and 4) repeat from 2. The merge order used (step 2) is of dire importance since it ultimately determines which states get tested for compatibility. The ALERGIA merge order (and also that of the MDI) is defined as follows: associate with each state of the PTA the (shortest) prefix that ends in that state. The states are then numbered according to the lexicographic ordering of their prefix strings. Other orderings are also possible, most notably the data- and evidence driven orderings.

What sets the two algorithms apart is the compatibility criterion that they use. ALERGIA assumes a Bernoulli distribution for each labelled outgoing edge (arc?). Compatibility is rejected if the confidence ranges between two states do not overlap for each symbol (label?) w.r.t. a pre-specified α.

This compatibility criterion is deficient in three ways. First, the Hoeffding bound is an asymptotic bound, and as such might be wholly inappropriate for finite (small) samples. Second, the outgoing labelled edges are tested independently of each other. And third, the prefix probability, or probability mass associated with the state is not taken into account.

The MDI algorithm on the other hand bases its compatibility test on the Kullback-Leibler divergence between pre- and post-merge automata. More precisely, compatibility is rejected if the ratio of the KL divergence increment and size decrement between pre- and post-merge automata is below the algorithm precision parameter. It is important to note that the KL divergence is between the p.d.f. over strings for both automata, not between the 'next symbol' p.d.f. of the merge candidates. This makes the MDI test a global compatibility test, as opposed to the local test of ALERGIA.

MAlergia [Kermorvant and Dupont, 2002], a variant of ALERGIA, also addresses these deficiencies, but does so with a local compatibility test. The Hoeffding bound is replaced with a multinomial test and a Fisher exact test is used when the amount of data is insufficient to validate the multinomial test.

Expectations

The results that I expect are that breaking up the PCFG rules, by either of above-mentioned methods, will have a beneficial effect on the parse performance of the resulting generalised grammar. To avoid having to do a full parse run to evaluate any changes applied during the experiments I will use the perplexity of both models on a held-out data set.

There are three points that will be considered a (minor) succes (each):

1. improved parse performance of the SFSM inferred grammar

2. lower perplexity of the SFSM model with respect to the n-gram model on the held out data for an equal amount of rules.
3. lower number of rules of the SFSM model for a similar level of perplexity

experiment setup and notation will be described in the next chapter.

3 METHODOLOGY

This chapter will give a more detailed description of the methods used.

3.1 NOTATION

MARKOV

The probability of a sequence of symbols w_1^n, with $w_i \in T$ for $1 \leq i < n$ and $w_n =$ stop, stop $\notin T$, is given by the product of the conditional probabilities of all the individual symbols:

$$P(w_1^n) = P(w_1) \cdot \prod_{i=2}^{n} P(w_i|w_1^{i-1}). \tag{3.1}$$

the probability of a symbol w_i is conditioned on the entire history w_1^{i-1} that precedes it. Making the assumption that the dependence of a word on symbols further back in its history is of such insignificance that it is negligible,

$$P(w_i) \approx P(w_i|w_{i-k}^{i-1}) \tag{3.2}$$

leads to the conditional probability model known as a Markov model of order k:

$$P(w_1^n) = P(w_1) \cdot \prod_{i=2}^{k} P(w_1|w_1^{k-1}) \cdot \prod_{i=k+1}^{n} P(w_i|w_1^{i-1}). \tag{3.3}$$

Where the (now reduced) conditional probabilities are estimated by the empirical distribution. The empirical distribution, is the relative frequency estimate given a large sample

$$P(w_i|w_{i-k}^{i-1}) = \frac{P(w_{1-k}^{i-1}w_i)}{\sum_{w_{i-k}^i} w_{i-k}^i} = \frac{P(w_{1-k}^{i-1}w_i)}{P(w_{1-k}^{i-1})}. \tag{3.4}$$

This is no different for the rules of the treebank grammar.

Stochastic Deterministic Finite Automata

A stochastic deterministic finite automata M is a tuple (Q, T, q_s, δ, p). The set T is the set of symbols, Q is the set of states, the transition function $\delta : Q \times T \rightarrow Q$ maps state, symbol pairs to states and finally the probability function $p : Q \times T \cup \{\text{stop}\} \rightarrow [0, 1]$. For $a \in T$ the probability function $p(q, a)$ is zero if $\delta(q, a)$ is undefined, $p(q, \text{stop}) = \sum_{a \in T} 1 - p(q, a)$ for all q.

The probability of a sequence of symbols w_1^n given the automata is the probability of the path through the automaton,

$$P(w_1^n) = \prod_{i=1}^{n} p(q_i, w_i) \cdot p(q_n, \text{stop}) \tag{3.5}$$

with the additional restriction that $q_1 = q_s$. $P(w_1^n)$ is zero if there is no such path.

In this work the probabilities assigned by p will also be derived from the relative frequencies. The counts are collected as follows. For an input sample I construct the weighted automaton known as the prefix tree acceptor. A weighted automaton is similar to the stochastic automaton, with the exception that the probability function p is replaced by a weight (mass?) function $m : Q \times T \rightarrow \mathbb{R}^+$. The PTA is converted into a SPTA with probability function p defined as:

$$p(q, a) = \frac{m(q, a)}{\sum_{t \in T} m(q, t)} = \frac{m(q, a)}{m(q)}, \tag{3.6}$$

where $m(q)$ is the weight of all the samples which have a prefix that passes through state q. In this way it is easy to see that the SPTA assigns a non-zero probability to words iff they are in the input sample.

Contrast this to the Markov models where this is generally not true. The n-gram models assign non-zero probability only to n-grams in the input sample. Words which do not occur in the input sample, but which consist of only n-grams we have seen will also get a non-zero probability.

Stochastic Deterministic Finite Grammars

Given an SDFA $M = (Q, A, \delta, q_0, p)$, then there exists a SDRG $G = (N, T, R, S, p)$ which has the same distribution language. The correspondence is as follows: For $1 \leq i, j \leq |Q|$, let there be a $N_i \in N$, $A = T$; for all transitions $\delta(q_i, a) = q_j$ exists a rule $R_i \rightarrow aR_j$, $S = N_0$ and $p^M(q_i, a) = p^Q(R_i \rightarrow aR_j)$. To be able to define the stop probabilities we need a rule $R_i \rightarrow \lambda$ for all states q_i which have a non-zero accept probability, i.e., $p^M(q_i, \text{stop}) = p^Q(R_i \rightarrow \lambda)$.

λ-removal for Extended SDRG

An extended SDRG G is a tuple (N, T, R, S, p), with N,T,S and p as for simple SDRGs and the rules R may have the forms:

$$A \rightarrow aB$$
$$A \rightarrow a$$
$$A \rightarrow \lambda$$

for $A, B \in N$ and $a \in T$.

An ESDRG-λ is an SDRG G for which $L(G)$ does not contain the empty string. λ-removal for ESDRG-λ proceeds as follows:

1. partition the non-terminals in three sets, based on their rhs:

   CLEAN no λ in rhs

   DIRTY only λ in rhs

   MIXED both with- and without λ in their rhs

2. for all non-terminals D in dirty, $p(D \rightarrow \lambda) = 1$

   a) for all C in clean s.t. $C \rightarrow aD$, let $d = p(C \rightarrow aD)$ and $e = p(C \rightarrow a)$. Remove $C \rightarrow aD$ and $C \rightarrow a$ from the grammar, add $C \rightarrow a$ with probability $p(C \rightarrow a) = d + e$.

   b) Likewise for M in mixed with $M \rightarrow bD$.

   Dirty is now empty. All productions $R \rightarrow aD$ involved in a derivation $\alpha R \Rightarrow \alpha a D \Rightarrow \alpha a \lambda \Rightarrow \alpha a$ now have the same yield, with derivation $\alpha R \Rightarrow \alpha a$ with the same probability.

3. for all M in mixed, 1) N in clean or mixed, with $N \rightarrow aM$, let $f = p(M \rightarrow \lambda)$, $g = \sum_{\alpha \neq \lambda} p(M \rightarrow \alpha) = 1 - f$ and $h = p(N \rightarrow a)$, $i = p(N \rightarrow aM)$. Set $p(N \rightarrow a) = h + if$, $p(N \rightarrow aM) = ig$, remove $M \rightarrow \lambda$ and renormalise all remaining $M \rightarrow \beta$.

   Pre-removal, the rule $M \rightarrow \beta$ can either be the final step in a derivation (when β is a or λ), or an intermediate step. The first case is the same as those for elements of dirty. In the second case the derivational probabilities from M onwards change by a multiplicative factor. This change is compensated for by reducing the probability of all rewriting rule which rewrite as M. There always is a previous rule because of the fact that the language does not contain the empty string (hence the start symbol never produces λ).

Entropy & Co.

The entropy of a probability distribution function P is defined as:

$$H(P) = \sum_x P(x) \log \frac{1}{P(x)} \tag{3.7}$$

$$= -\sum_x P(x) \log P(x) \tag{3.8}$$

The Kullback-Leibler divergence or relative entropy:

$$D(P, Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)} \tag{3.9}$$

the relative uncertainty of Q w.r.t. P.

The cross-entropy:

$$H(P, Q) = \sum_x P(x) \log \frac{1}{Q(x)}$$
$$= H(P) + D(P, Q)$$

uncertainty of Q w.r.t. P, note that the entropy is the cross-entropy of P w.r.t. itself.

The perplexity is closely related to the cross-entropy since it measures how 'surprised' a model is when presented with a some data. The discrepancy between the probability assigned by the model and the empirical distribution as implied by the data (which for large sets will be a close approximation of the real distribution) is then used as a measure of the fit of the model to the data.

$$\text{Perplexity}_{LM}(S) = 2^{H(P(S), P_{LM}(S))}$$

3.2 Data Split

I will not use the train/development/test/validation split that is common for the WSJ. The common setup is the following, use sections two through 22 (21?) as training data, section one as development test data and 23 as validation test data. Instead i will perform the ten-fold cross-validation (over the sections two through 22) that is more common in the machine learning field.

The dataset is segmented into the ten folds at the sentences level. To be more precise, the dataset is first read in, then the sentences are shuffled and finally split into ten equal subsets (give or take a single sentence).

3.3 Data Preprocessing

The WSJ data is preprocessed to produce two different train sets per fold. First, for both models, similar to Charniak [1996], the functional tags are stripped, null and empty elements are removed; and second, head annotation is added (using the rules described in Collins [1999].)

I then extract the grammar and lexicon from both datasets and further process either independently of the other.

Lexicon

The lexicon is a multiset of tuples (word, POStag). I assume that the set of POStags is a known, closed set of symbols, irrespective of the size of the train set. The set of words on the other hand is not treated specially. This implies that there are two types of problematic cases (as far as the lexicon is concerned.) we might encounter during parsing. Both are tuples we have not seen during training:

- words we have seen, but not with the correct POStag (the tag with which it occurs in the test set.)
- words which do not occur in the train data at all.

I simply collect the statistics for rare words in the training set, here rare words are words that occur at most five times, and use the resulting tag distribution for unknown words. Subsequently i do add-one smoothing over all tuples.

Grammar

Starting off with a short overview of the models i will consider:

BASELINE identity transform, using the literal rules as extracted from the treebank

BIGRAM BASELINE identity transform, the rule righthand-sides are Markovised with a bigram model.

HEAD BASELINE head annotation, straight rule extraction.

HEAD BIGRAM head annotation, bigrams over righthand-sides, head-outwards

HEAD SFSM head annotation, automata inferred over righthand-sides with varying parameter settings, head-outward and left-to-right.

The undecorated treebank trees are only used to provide a baseline for the other models.

Headless Baseline

Given the tree in figure 3.1, we extract the following grammar for the headless baseline:

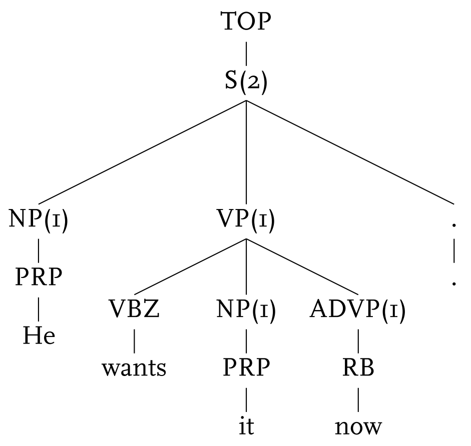


Figure 3.1: Example tree. The numbers are the indices of the head child of each node.

1	TOP	→	S
1	S	→	NP VP .
2	NP	→	PRP
1	VP	→	VBZ NP ADVP
1	ADVP	→	RB

all the terminals (the words) and pre-terminals (the POStags) end up in the lexicon.

Headless Bigrams

Generating and n-gram model for the grammar rules is not very different from generating it for any other set of samples. Converting the n-gram statistics (back) into rules which can be used by the parser will, on the other hand, involve an extra step. There are at least two ways to perform this step. One is to explicitly write out the rules with their associated probabilities. This is the simpler of the two methods, but becomes problematic if the Markov chain has cycles (which is virtually unavoidable.) Going down this route would require an explicit limit on the rule count, a limit on the length or probability mass, both are possible.

The other is to convert the implied FSM into a right- or left linear grammar. This way we can maintain an exact representation of the underlying Markov model, even for 'infinite' sequences. A minor inconvenience presents itself when we condition on specific contexts (lhs of the rule or the head tag.) To avoid unwanted interference between different Markov models we need to ensure that internal nodes, or non-terminals, of the partial subgrammars are unique. (The subgrammars are partial in the sense that the yield of the subgrammars are sets of phrases, strings of non-terminals and terminals.)

This results in the following grammar:

1	TOP	$\rightarrow$	S
1	S	$\rightarrow$	NP S1
1	S1	$\rightarrow$	VP S2
1	S2	$\rightarrow$	.
2	NP	$\rightarrow$	PRP
1	VP	$\rightarrow$	VBZ VP1
1	VP1	$\rightarrow$	NP VP2
1	VP2	$\rightarrow$	ADVP
1	ADVP	$\rightarrow$	RB

the first thing that jumps out is that the number of rules has *increased*, and so has the number of grammar symbols. This is not as bad as it may seem at first. The increase gets proportionally smaller with an increasing amount of rules (unless these rules share no commonality between one another.) Furthermore, because the grammar that the parser uses for actual parsing is the, binarised, Chomsky Normal Form of whatever grammar it is fed, this means that any rules with rhses longer that two symbols will necessarily be broken up before parsing.

Also clearly visible are the subgrammars, one for each original non-terminal.

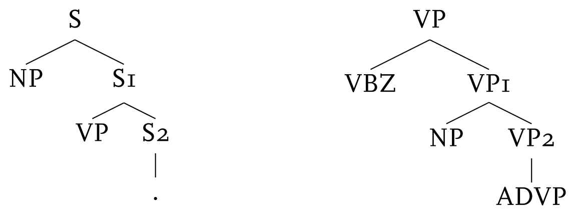


Figure 3.2: Parsing the original rules. The root of the first subtree is the original non-terminal 'S', and the fringe of the tree is the original rule 'NP VP .', similarly for the other subtrees.

The SFSM case is treated in the exact same way.

Dealing with Heads

Phrases are centred around their head [Radford, 1988], this has prompted researchers in computational linguistics to do head-outwards generation rules. Even though for Markov models it does not matter in which direction a sequence is processed, the count of n-grams does not change, therefore the inferred probabilities do not change. For SFSMs this does matter, since the ordering of the sequences typically influences the order in which merges are considered.* In addition to that, for deterministic automata, the extra determinacy constraint imposes additional restrictions on the part of the search space that can be explored.

I will use the same notation as in [Collins, 1999]. After enriching the tree with all the head tags, the subtrees with root P and head h have the following structure:

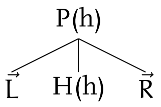


Figure 3.3: head enriched subtree rooted at P, with head child H, head tag h and left- and right modifiers $\vec{L}$ and $\vec{R}$

This leads to the following collection of rules (see figure 3.3):

Pi	$\rightarrow$	P(h)
P(h)	$\rightarrow$	P-L H(h) P-R
P-L	$\rightarrow$	Left partial subgrammar
P-R	$\rightarrow$	Right partial subgrammar

where the sequences of left L_l^n and right modifiers R_r^m can consist of zero or more symbols.

One way of dealing with heads is to add the head information, POStag symbol, directly to the non-terminal, thereby expanding the non-terminal set considerably. This has two direct consequences, one, it allows a much more specific assignment of probabilities to differently distributed tags. Two, it exacerbates the data sparsity problem. Not being convinced about the effectiveness of the former and slightly troubled by the latter I chose not to deal with head information in this way.

Only the head child retains the attached head tag, the left- and right sequences have their head tags stripped and intermediate symbols are introduced (denoted by the lowercase i suffix). A set of unary rewrite rules is introduced to allow the intermediate nodes to select a head.

Head Baseline

Grouping the collections of rhses by lhs and the head produces four cases which will be dealt with separately. The cases are: empty left and right modifiers, empty left modifier and non-empty right modifier, non-empty left- and empty right modifier and finally both left- and right modifiers non-empty. In this way I can avoid

*this happens to be the case for the lexicographic ordering used by the MDI algorithm the problem of (generating) empty rules when either sequence of dependent is empty.

1	TOP	$\rightarrow$	Si
1	Si	$\rightarrow$	S-VBZ
1	S-VBZ	$\rightarrow$	S-VBZ-L VP-VBZ S-VBZ-R
1	S-VBZ-L	$\rightarrow$	NPi
1	S-VBZ-R	$\rightarrow$	.
2	NPi	$\rightarrow$	NP-PRP
2	NP-PRP	$\rightarrow$	PRP
1	VP-VBZ	$\rightarrow$	VBZ VP-VBZ-R
1	VP-VBZ-R	$\rightarrow$	NPi ADVPi
1	ADVPi	$\rightarrow$	ADVP-RB
1	ADVP-RB	$\rightarrow$	RB

One undesirable side effect of this method (scheme?) is that it assumes that the sequences of left- and right modifiers are dependant only on the lhs (parent non-terminal), and that they are independent of one another. The experiments will have to show whether this is a valid assumption to make.

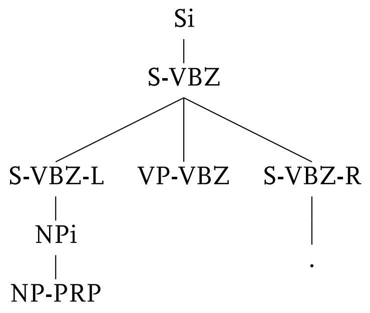


Figure 3.4: Rebuilding the original rule with head information.

3.4 Parsing

The parser used for all experiments is BitPar [Schmid, 2004]. BitPar is a highly optimised chart parser for PCFGs. Given a grammar, a lexicon and a text to parse it computes the entire parse forest for that sentence and outputs the most probable (Viterbi) parse, the top n most probable parses or the entire parse forest (given enough disk space).

This has two consequences, the most probable parse, as returned by the parser, really is the most probably parse and not an approximation and two, comparatively long parse times.[†]

Unknown words can be handled in two ways. The first way to handle them is to supply a file with possible POStags for unknown words and the second way is to provide a SFST to map words to tags. We choose the former for the sake of simplicity. Additionally BitPar has the option to smooth the word classes, the smoothing it uses is simple add-δ or additive smoothing. BitPar automatically discards POStags which are less likely than some fraction of the most likely POStag. This can be set via a command line option.

3.5 Post-processing

As described above, all the models involve one or more transformations on the original treebank and since we want to compare parse performance relative to the original forms, we will have to undo any transformations that may have been applied.

In my case that means one of the following:

- low level, character escaping due to parser peculiarities
- delinearising subgrammars
- stripping head information

The first, low level, inverse transform is a simple textual substitution and will not be described any further here.

Delinearising the various subgrammars is a matter of walking through the generated tree collapsing ‘expansion nodes’. Expansion nodes are nodes for which the rightmost child is an intermediate node. Collapse the right branching structure under the expanded node by collating all siblings of the intermediate nodes, in order, all the way up to the first non-intermediate descendant node. Also append this non-intermediate node to the list. Finally, replace the children of the expansion node with the nodes in the generated list.

Undoing the head transformation is very similar. For each node, if it has intermediate children, replace its children by all its grandchildren (again, in order). Afterwards, map all head annotated symbols back onto the original forms.

[†]just under five seconds per sentence on a 2.8GHz P4 for a 18,500 rule grammar with 4800 symbols.

3.6 Evaluation

Perplexity

A problem with the perplexity as a measure of fit is that it is undefined if the model under consideration assigns zero probability to an event in the dataset. The common solution is to mix, or interpolate, the model that one is testing. Either with the target distribution or with a probability model which is guaranteed to assign non-zero probability to all events. Specifically, a unigram model does *not* have this property if not all symbols in the test data occur in the train data. I therefore interpolate with a uniform distribution over all possible symbols.

The other approach has been taken by Infante-Lopez [2005]. Which is to ignore all problematic cases. The benefits of this methods are that the results are not muddled by some inappropriate distribution. On the other hand, it skews the results markedly in favour of the, for the dataset ill-defined, model.

During preliminary testing a disconcerting fact came up. The induced SFSMs were unable to improve on the scores obtained by the n-gram models. At best they were only able to match the perplexity values. The handful of tags that did favour the SFSMs were rare, both in the test as well as in the train data, and as such, had a negligible effect of the resulting grammar.

Parsing

There are a couple of historical facts that have an artificially inflating effect on all published scores regarding the parse scores. These are, in no particular order:

1. limiting parsing to sentences of length 40 or less.
2. treating some tags as equivalent (ADVP and PRT).
3. ignoring all (non-sentence final) punctuation.

The first point is entirely a problem of the past. In the early nineties computing power was considerable, but nowhere near the levels they are today. As the sentence length increases so does parse ambiguity and hence parsing time. Nowadays this is not a problem anymore for PCFG parsers (it still is for the more powerful models though), but the numbers reported usually still are for the, better scoring, length limited sentences. The difference between including the scores for all sentence lengths and those of length 40 or less is about 1.0–1.5 points of FMeasure.

The second point is more debatable, but its effect is negligible, less than 0.1 points of FMeasure.

The final point is similar to the previous one, but has had a different reasoning and a significantly greater impact. The reason people in the parsing community ignore punctuation is that humans tend to be very inconsistent in their use of punctuation. Probably because it is only a hint and not a hard rule to help the reader break up longer passages, which would have been broken up by pauses in spoken language. Ignoring punctuation lead to a 1.5–2.5 points increase.

It is my view that a if a relabelling is performed, then it should be applied to the data in a preprocessing step and not as an afterthought hack in the evaluation. The adding or removing of nodes is a more drastic measure and a more well thought out reason for ignoring it should be presented.

The major exception is the exclusion of the brackets/parentheses labelled TOP (or ROOT). These were added to the treebank to ensure that each trees has a single root element (and thus that the grammar has a single unique start symbol.)

4 Results

This chapter an analysis of the results of the experiments. As explained in the previous chapter, all reported results are for all sentence lengths. No labels (punctuation and the like) are ignored or removed apart from the TOP label.

A concise overview of all results can be found in table 4.1.

4.1 Overall

Baseline

The baseline model is the simple treebank grammar which was obtained by simple extraction of the PCFG rules as they occurred in the corpus. This experiment has been included to a) put the results in this thesis in perspective to the results generally reported in the literature, and b) to form the baseline for the next model.

For the sake of completeness, the results for the simple PCFG for sentences up to length 40 are 66.87 LR (σ 0.37) and 71.99 LP (σ 0.35).

Baseline with Head Annotation

The model in this experiment has the head information added as described in section 3.3, but is otherwise identical to the baseline PCFG.

Results for sentences up to 40 words are 68.11 LR (σ 0.20) and 73.11 (σ 0.21).

Comparing the results of the two models shows that head enriching the trees and breaking up the PCFG rules helps both labelled recall and -precision.

4.2 Unseen Rules

A closer inspection reveals that 833 of the total of 3894 test sentences, approximately 21%, of the first fold contain at least one unseen rule. The parse results for those sentences are shown in table 4.2. The average sentence length for this

Table 4.1: Parse results for all sentence lengths, μ is the mean, σ the standard deviation.

model	LR		LP		FMeasure		CB	
	μ	σ	μ	σ	μ	σ	μ	σ
baseline	65.69	0.31	70.73	0.31	68.12	0.29	3.21	0.06
baseline w/ head	66.83	0.23	71.72	0.23	69.19	0.22	3.02	0.05
bigram	65.90	0.21	71.49	0.17	68.58	0.18	3.02	0.04
n-gram	66.78	0.23	71.98	0.33	69.28	0.23	2.99	0.08
fsm 8e-5	63.57	0.35	63.95	0.41	63.76	0.34	3.90	0.06
fsm var	60.50	0.56	63.07	0.58	61.76	0.57	4.05	0.10

sample is rather high, just over 30 instead of the average of 22 for the entire WSJ. The numbers are for a single fold only, hence the lack of a standard deviation.

We see that the baseline is severely affected, dropping almost six points of FMeasure. The FSM method is consistently catastrophic, not even starting out with a great margin of error could help the eight points drop in performance, the relative error increase is 17.8% versus 14.6% for the baseline.

Remarkably, the results for the bigrams and the variable order n-grams are identical to each other. Their drop is not as drastic as either the FSM or the baseline, about two points; this makes their relative error increase less than half of the other two methods (the error increases are 4.5% and 6.9% for the bigram and variable order n-gram respectively.)

Table 4.2: Parse results for sentences with unseen rules, single fold

model	LR	LP	FMeasure	CB
baseline	60.69	63.74	62.18	4.76
bigram	66.22	68.12	67.16	4.22
n-gram	66.22	68.12	67.16	4.22
fsm 8e-5	57.22	55.54	56.36	6.03
fsm var	53.45	54.17	53.81	6.17

5 Conclusion

Assigning proper probabilities to grammars rules turns out to be harder than it may seem at first. The primary goal of this thesis was to be able to efficient and effective way to assign these probabilities. And in doing so, be able to dispense with the ubiquitous n-grams in another part of natural language processing.

This thesis gives inconclusive evidence as to whether the more powerful regular grammars are unable to outperform the simpler Markov models. Using variable context lengths, provides a small but real increase in the performance of said Markov models. Why then do the finite state machines not have the same benefits? A possible but unlikely answer would be that the inference algorithm is unable to identify the 'correct' automaton. The MDI algorithm has been successfully applied to artificial and real problems [Thollard et al., 2000].

Another option that has not been ruled out is the actual implementation, which might still be flawed.

5.1 Revisiting Thesis Statement

The questions posed at the start of the thesis were:

1. How can we sensibly generalise treebank grammars?
2. Can we escape the clutches of the omnipresent n-grams?

The first has remained largely unanswered. Yes, you can use finite state automata, but it is not a particularly good idea. And the second has been a resounding no. There is more to be gained from using as much context as one can get from a treebank and subsequently applying better informed smoothing methods. Ones which, unlike the Markov models, can interpolated or back-off to a coarser model.

Bibliography

The Computational Analysis of English: A Corpus-based Approach. Longman Publishing Group, Harlow, UK, dec 1987. ISBN 0582291496.

Speech and Natural Language, Proceedings of a Workshop held at Pacific Grove, California, USA, February 19-22. 1991, 1991. Morgan Kaufmann. ISBN 1-55860-207-0.

Naoki Abe and Manfred Warmuth. On the computational complexity of approximating distributions by probabilistic automata. In *Proceedings of the Third Workshop on Computational Learning Theory*, pages 52–66. Morgan Kaufmann, 1990. URL `citeseer.ist.psu.edu/abe90computational.html`.

S. Abney, R. Schapire, and Y. Singer. Boosting applied to tagging and PP attachment, 1999. URL `citeseer.ist.psu.edu/abney99boosting.html`.

Dana Angluin. Inference of reversible languages. *J. ACM*, 29(3), 1982. ISSN 0004-5411.

Daniel M. Bikel. Intricacies of collins' parsing model. *Computational Linguistics*, 30(4):479–511, dec 2004. ISSN 0891-2017. doi: http://dx.doi.org/10.1162/0891201042544929.

Ezra Black, Steven P. Abney, D. Flickenger, Claudia Gdaniec, Ralph Grishman, P. Harrison, Donald Hindle, Robert Ingria, Frederick Jelinek, Judith L. Klavans, Mark Liberman, Mitchell P. Marcus, Salim Roukos, Beatrice Santorini, and Tomek Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of english grammars. NAA [1991]. ISBN 1-55860-207-0. URL `http://acl.ldc.upenn.edu/H/H91/H91-1060.pdf`.

Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, May 1973.

J. Bresnan, R. M. Kaplan, S. Peters, A, and Zaenen. Cross-serial dependencies in dutch. In FormalComplexityNaturalLanguage87 FormalComplexityNaturalLanguage87.

Peter F. Brown, Peter V. de Souza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4), December 1992. URL citeseer.ist.psu.edu/article/brown92classbased.html.

Rafael Carlos Carrasco Jiménez and Jose Oncina Carratalá. Learning stochastic regular grammars by means of a state merging method. In *ICGI 94: Proceedings of the Second International Colloquium on Grammatical Inference and Applications*, volume 862, pages 139–152, London, UK, 1994. Springer-Verlag. ISBN 3-540-58473-0. URL citeseer.ist.psu.edu/carrasco94learning.html.

Rafael Carlos Carrasco Jiménez and Jose Oncina Carratalá. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO (Theoretical Informatics and Applications)*, 33(1):1–20, 1999.

Eugene Charniak. Tree-bank grammars. In *AAAI/IAAI, Vol. 2*, pages 1031–1036, 1996. URL citeseer.ist.psu.edu/charniak96treebank.html.

Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. Technical Report TR-10-98, Computer Science Group, Harvard University, august 1998.

Noam Chomsky. Three models for the description of language. *IRA transactions on information theory*, 1956.

Alexander Clark and Frank Thollard. PAC-learnability of probabilistic deterministic finite state automata. *Machine Learning Research*, pages 473–497, 2004.

Michael John Collins. A new statistical parser based on bigram lexical dependencies. In Arivind Joshi and Martha Palmer, editors, *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, pages 184–191, San Francisco, 1996. Morgan Kaufmann Publishers. URL citeseer.ist.psu.edu/article/collins96new.html.

Michael John Collins. *Head-Driven Statistical Models for Natural Language Parsing*. PhD thesis, University of Pennsylvania, 1999.

Micheal John Collins. Three generative, lexicalized models for statistical parsing. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 16–23, Somerset, New Jersey, 1997. Association for Computational Linguistics. URL citeseer.ist.psu.edu/article/collins97three.html.

Colin de la Higuera and Franck Thollard. Identification in the limit with probability one of stochastic deterministic finite automata. In Oliveira [2000]. ISBN 3-540-41011-2.

François Denis. Learning regular languages from simple positive examples. *Machine Learning*, 44(1/2):37–66, 2001.

François Denis and Yann Esposito. *Learning Theory*, volume 3120 of *Lecture Notes in Computer Science*, chapter Learning Classes of Probabilistic Automata, pages 124–139. Springer Berlin/Heidelberg, june 2004.

Jason Micheal Eisner. *Smoothing a Probabilistic Lexicon via Syntactic Transformations*. PhD thesis, University of Pennsylvania, 2001.

Jean-Babtiste Estoup. Les gammes stenographiques. *Institut Stenographique de France*, 1916.

FormalComplexityNaturalLanguage87. *The Formal Complexity of Natural Language*. Reidel, Dordrecht, Nederland, 1987.

King Sun Fu and T. Huang. Stochastic grammars and languages. *International Journal of Computer and Information Sciences*, 1(2):135–170, 1972.

Daniel Gildea. Corpus variation and parser performance. In Lillian Lee and Donna Harman, editors, *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, pages 167–202, 2001.

Mark E. Gold. Language identification in the limit. *Information and Control*, 10(5): 447–474, 1967. URL http://www.isrl.uiuc.edu/~amag/langev/paper/gold67limit.html.

I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika*, 40(3):237–264, 1953. doi: doi:10.2307/2333344.

Theodore Edward Harris. *The Theory of Branching Processes*, volume 119 of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete*. Springer-Verlag, Berlin, 1963.

J. C. Henderson and E. Brill. Bagging and boosting a treebank parser. *ArXiv Computer Science e-prints*, 2000.

John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979. ISBN 0321462251.

Norbert Hornstein and David Lightfoot, editors. *Explanation in Linguistics: The Logical Problem of Language Acquisition*, pages 9–31. Volume unspeakabledepths of , Hornstein and Lightfoot [1981b], 1981a.

Norbert Hornstein and David Lightfoot, editors. *Explanation in Linguistics: The Logical Problem of Language Acquisition*. Longman, 1981b.

Sandra E. Hutchins. *Stochastic sources for context-free languages*. PhD thesis, Department of Applied Physics and Information Science, University of California, San Diego, 1970.

Gabriel Gaston Infante-Lopez. *Two-level Probabilistic Grammars for Natural Language Parsing*. PhD thesis, Universiteit van Amsterdam, 6April 2005.

Mark Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4), 1998. URL citeseer.ist.psu.edu/article/johnson98pcfg.html.

Christopher Kermorvant and Pierre Dupont. Stochastic grammatical inference with multinomial tests. In *LNAI*, volume 2484, pages 149–160. Springer-Verlag, 2002.

Dan Klein and Christopher D. Manning. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics*, pages 423–430, Morristown, NJ, USA, 2003. Association for Computational Linguistics. doi: http://dx.doi.org/10.3115/1075096.1075150.

David M. Magerman. Statistical decision-tree models for parsing. In *Meeting of the Association for Computational Linguistics*, pages 276–283, 1995. URL citeseer.ist.psu.edu/magerman95statistical.html.

Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, second edition, 1999. ISBN 0-262-13360-1.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1994. URL citeseer.ist.psu.edu/article/marcus93building.html.

Maurice Nivat and Andreas Podelski. Minimal ascending and descending tree automata. *SIAM J. Comput.*, 26(1):39–58, 1997.

Arlinedo L. Oliveira, editor. *Grammatical Inference: Algorithms and Applications, 5th International Colloquium, ICGI 2000, Lisbon, Portugal, September 11–13, 2000, Proceedings*, volume 1891 of *Lecture Notes in Computer Science*, 2000. Springer. ISBN 3-540-41011-2.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia, July 2006. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P06/P06-1055`.

Carl Pollard and Ivan A. Sag. *Head-Driven Phrase Structure Grammar*. University of Chicago Press and CSLI Publications, Chicago, Illinois, 1994. URL `citeseer.ist.psu.edu/pollard94headdriven.html`.

W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, 1988.

Andrew Radford. *Transformational Grammar: A First Course*. Cambridge University Press, Cambridge, UK, first edition, may 1988. ISBN 0521347505.

Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 1996. URL `citeseer.ist.psu.edu/article/ron96power.html`.

Dana Ron, Yoram Singer, and Naftali Tishby. On the learnability and usage of acyclic probabilistic finite automata. *Journal of Computer and System Sciences*, 56 (2):133–152, 1998. URL `citeseer.ist.psu.edu/ron95learnability.html`.

Ronald Rosenfeld. Two decades of statistical language modeling: Where do we go from here? In *Proceedings of the IEEE*, volume 88, pages 1270–1278. IEEE, aug 2000.

Ivan A. Sag, Thomas Wasow, and Emily M. Bender. *Syntactic Theory: A Formal Introduction*. CSLI Lecture Notes. Center for the Study of Language and INF, 2ed edition, 2003. ISBN 1575864002.

Geoffrey Sampson. *The grammatical database and parsing scheme*, chapter 7, pages 82–96. 1987.

Remko J.H. Scha. Taaltheorie en taaltechnologie: competence en performance. In Q.A.M. de Kort and G.L.J. Leerdam, editors, *Computertoepassingen in de Neerlandistiek*, LVVN-jaarboek, pages 7–22. Landelijke Vereniging van Neerlandici, 1990.

Helmut Schmid. Efficient parsing of highly ambiguous context-free grammars with bit vectors. In YoMomma [2004].

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

S. M. Shieber. Evidence against the context-freeness of natural language. In FormalComplexityNaturalLanguage87 FormalComplexityNaturalLanguage87, pages 320–334.

Thomas A. Sudkamp. *Languages and machines: an introduction to the theory of computer science*. Addison-Wesley, 2nd edition, 1998.

Franck Thollard, Pierre Dupont, and Colin de la Higuera. Probabilistic DFA inference using Kullback-Leibler divergence and minimality. In *Proc. 17th International Conf. on Machine Learning*, pages 975–982. Morgan Kaufmann, San Francisco, CA, 2000. URL `www.sfs.nphil.uni-tuebingen.de/~thollard/Recherches/Icml2k/icml2k.html`.

Leslie G. Valiant. A theory of the learnable. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 436–445, New York, NY, USA, 1984. ACM Press. ISBN 0-89791-133-4. doi: http://doi.acm.org/10.1145/800057.808710.

José Luis Verdú Más, Mikel L. Forcada Zubizarreta, Rafael Carlos Carrasco Jiménez, and Jorge Calera Rubio. Tree k-grammar models for natural language modelling and parsing. *0302-9743 - Lecture Notes in Computer Science - Lecture Notes in Artificial Intelligence*, pages 53–63, 2002.

Charles S. Wetherell. Probabilistic languages: A review and some open questions. *ACM Computing Surveys*, 12(4):361–379, 1980. ISSN 0360-0300. doi: http://doi.acm.org/10.1145/356827.356829.

YoMomma, editor. *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

George Kingsley Zipf. Selective studies and the principle of relative frequency in language. Harvard University Press, Cambridge, MA, 1932.