# A Corpus-Based Approach
# to
# Language Learning

Eric Brill

A Dissertation

in

Department of Computer and Information Science

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

1993

Mitchell Marcus

Supervisor of Dissertation

Mark Steedman

Graduate Group Chairperson

# Acknowledgements

Many people deserve thanks in helping me progress from my mother's womb to finishing my dissertation. Rather than beginning with the doctor who delivered me and filling pages and pages with names, I'll say one big generic thanks to all who deserve to be thanked but aren't mentioned here by name.

My advisor, Mitch Marcus, deserves a great deal of thanks for his constant support and encouragement. His enthusiasm helped make my research a lot more fun.

I thank my committee (Steve Abney, Lila Gleitman, Aravind Joshi and Mark Liberman) for their feedback, which was significant in shaping my dissertation. I also thank them for allowing me to graduate.

I believe the men and women who have sacrificed over the years to defend our Bill of Rights and Constitution deserve special thanks, for without their commitment, freedom of thought and expression might not exist.

My parents instilled in me a love for education from an early age, which provided the driving force for me to pursue a degree, as well as the belief that one should always doubt, question, and learn.

I met Meiting Lu when I first arrived at the University of Pennsylvania. She managed to keep me sane during my 3.75 years at Penn. It is Meiting and her friendship that deserve the biggest thanks.

# Abstract


A CORPUS-BASED APPROACH TO LANGUAGE LEARNING

Eric Brill

Supervisor: Mitchell Marcus

One goal of computational linguistics is to discover a method for assigning a rich structural annotation to sentences that are presented as simple linear strings of words; meaning can be much more readily extracted from a structurally annotated sentence than from a sentence with no structural information. Also, structure allows for a more in-depth check of the well-formedness of a sentence. There are two phases to assigning these structural annotations: first, a knowledge base is created and second, an algorithm is used to generate a structural annotation for a sentence based upon the facts provided in the knowledge base. Until recently, most knowledge bases were created manually by language experts. These knowledge bases are expensive to create and have not been used effectively in structurally parsing sentences from other than highly restricted domains. The goal of this dissertation is to make significant progress toward designing automata that are able to learn some structural aspects of human language with little human guidance. In particular, we describe a learning algorithm that takes a small structurally annotated corpus of text and a larger unannotated corpus as input, and automatically learns how to assign accurate structural descriptions to sentences not in the training corpus. The main tool we use to automatically discover structural information about language from corpora is transformation-based error-driven learning. The distribution of errors produced by an imperfect annotator is examined to learn an ordered list of transformations that can be applied to provide an accurate structural annotation. We demonstrate the application of this learning algorithm

to part of speech tagging and parsing. Successfully applying this technique to create systems that learn could lead to robust, trainable and accurate natural language processing systems.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Structural Descriptions and Language Learning

## 1.1 Structural Information and Natural Language

Part of a person's knowledge of language consists of knowing how to assign an abstract structural description to sentences. Included in this knowledge is an awareness of the word and phrase classes of a language, the members of each class, and the relationships that hold between classes. For instance, although an English speaker may not be aware of the linguistic labels, he is tacitly aware of more than just the linear structure of the sentence: *The boys eat.* Figure 1.1 shows some of the structural information tacitly known by English speakers about this short sentence. English speakers know that *eat* subcategorizes for a noun phrase that is not third person singular, that *boys* is the plural form of *boy*, that *boy* is a noun, that the two words *the boys* form a noun phrase, and that the three words *the boys eat* constitute a sentence.

Of the classes and relationships that hold in a language, some are superficial. Their existence appears to be fairly transparent, either because the classes roughly follow from well understood semantics or because they are syntactically surface-apparent. In other words, these classes and relationships can be described to some extent without recourse to deep semantic analysis and without the need to make reference to a detailed and abstract structural description. The annotated information in figure 1.1 is all fairly superficial. For

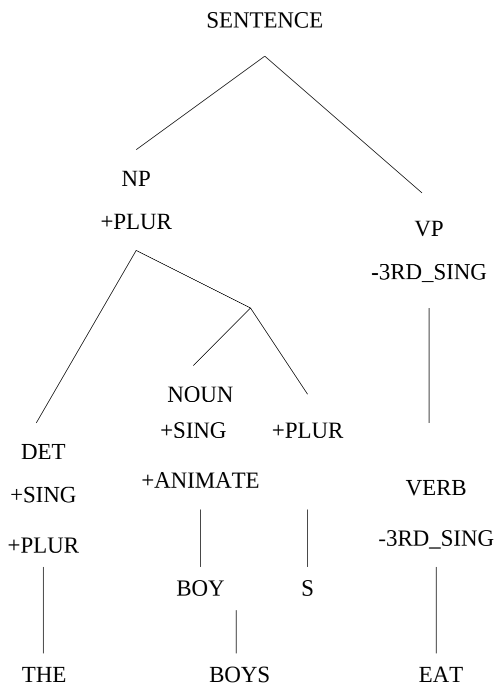



Figure 1.1: Structural Information.

example, although not absolutely correct, the semantic definition *a noun is a person, place or thing* is a fair criterion for membership into the class of nouns, or at least a basis from which we can understand the origination of the class.

However, there are a number of classes and relationships whose existence is not transparent. Pinker [90] discusses the class of dativizable verbs. In (1b), the verb *gave* has undergone dativization.

(1a) John gave a painting to the museum.

(1b) John gave the museum a painting.

Note that although *donated* is semantically very similar to *gave*, *donated* cannot undergo dativization.

(2a) John donated a painting to the museum.

(2b) * John donated the museum a painting.

People are able to ascertain whether a verb that they have never heard dativized can undergo dative shift. This productivity indicates that it cannot merely be that people assign words to the class of dativizable verbs when they see an example of the verb being dativized. Pinker argues that some very subtle semantic properties determine which verbs belong to this class.

In addition to subtle word classes, research in modern syntax has uncovered many nonsuperficial restrictions on what structural relationships can hold in a sentence. As one example of a relationship that is not surface-apparent, let us examine the that-trace effect[76]. How can we characterize what it is that permits sentences (3a) - (3c), but does not permit (3d)?

(3a) What do you think John likes?
(3b) What do you think that John likes?
(3c) What do you think fell?
(3d) * What do you think that fell?

One explanation syntacticians offer for this phenomenon involves the assumption that wh-words move to the beginning of a sentence to form a question, and leave an invisible trace in the position from which they moved. So, the question (3a) is formed from the base sentence *You think John likes what?*. When *what* moves to the front of the sentence, it leaves a trace. Traces are only allowed to appear in restricted positions in a sentence. They can only appear in positions where they can be governed by a real word. In order to be governed, a fairly complex structural relationship must hold between the trace and the word that governs it. This relationship holds in (3a) - (3c), but does not hold in (3d).

In this work we will concentrate on automating the learning of superficial structural information.[1] We would like to determine to what extent information such as the part of speech of a word in a particular context and skeletal phrase structure of sentences can be discovered automatically. If a complex description is needed to fully explain a phenomenon, we can ask to what extent the phenomenon can be explained or captured by a simple analysis of surface structure. For example, there are certainly cases where

[1] See [21] for an example of using information-theoretic corpus-based techniques to learn more complex phenomena that are not surface-apparent. In particular, we discuss methods of setting the V2 word-order parameter using distributional measures on a corpus with no structural annotation and beginning with only minimal assumptions about knowledge of language prior to learning.

the proper part of speech tag for a word in a context depends on deep analysis. We can quantify the extent to which this is the case, or at least find a lower bound, by building a program which tags using simple surface-structure information, and then checking how well such a tagger can perform on natural text.

Although the phenomena we wish to explore do not comprise the whole language picture, they do comprise a significant and interesting portion of it. Much of language understanding involves mastering the many superficial, but highly idiosyncratic, rules of the language. As testament to the vastness of superficial knowledge, Quirk and his colleagues [42] have compiled a 1779 page reference book entitled "A Comprehensive Grammar of the English Language" which records superficial facts about English, and even this book is no doubt incomplete.[2]

## 1.2 Understanding Human Language Learning

Since the work presented in this report and the research program of generative grammarians (e.g. [35]) share as a primary goal an explanation of how language can be learned, it is important to be clear about the differences between the two approaches. Although both approaches address language learning, the focus of the two approaches is quite different. Generative grammarians are interested in exploring the nature of language by uncovering language universals, properties held by all natural languages. The search for universals is of interest because it is a step towards differentiating the essence of natural language from the idiosyncrasies of any particular language. The search for universals may also provide an explanation for how a child learns language. It is now commonly believed that language learning cannot proceed as a purely inductive process with no a priori knowledge of the target grammar. Some of the roots of this belief (outlined in [34]) include:

1. Poverty of the stimulus: the quantity and quality of evidence in the environment is not conducive to learning.

2. Complex, non-surface-apparent grammatical constraints seem unlearnable.

[2] Of course, the reason for the vastness of this book could be attributed to the failure of the authors in finding the true, concise description of language.

3. Some knowledge of language appears to be shared by many diverse languages.

The Principles and Parameters approach [35] was offered as a model of how a child could come to acquire the skills which allow her to use language productively. In this model, language is divided into two parts: core and periphery. The periphery contains information which must be learned and is unique to a particular language, such as irregular morphology and idioms. The core contains innate linguistic universals. To account for the differences between languages, some of the rules (or constraints) in the core are parameterized. One such rule is pro-drop [63]. In English, the subject of a sentence is necessary in all but imperative sentences; in Spanish, the subject is optional. To account for this, people working under the Principles and Parameters model assume that there is an innate constraint which is underspecified, stating:

In your language, you { *can/cannot* } drop the subject of a sentence.

To learn a language, one must learn the peripheral facts and must find the proper settings of all parameterized core constraints. The Principles and Parameters model accounts for the ability to learn language despite the poverty of the stimulus, because one simple sentence could act as the trigger for properly setting the parameter of a complex rule. It also explains how complex language constraints can come to be known: these constraints are innate and need not be learned. One weakness of this approach is that in its current form it does not lend itself to algorithmic implementation.[3]

In this dissertation we also explore language learning, but we are addressing a different problem. The focus of our research is to find algorithmic approaches that are successful at learning information necessary to allow for the accurate and productive[4] structural analysis of a sentence. In a sense, we are empirically investigating whether the poverty of stimulus argument applies to learning the superficial phenomena investigated in this thesis. While people studying Principles and Parameters are exploring what facts about language could be accounted for by innate linguistic constraints, we are setting out to explore what facts about language are learnable by applying a learning algorithm to a

[3]This is not entirely true. [43] describes one attempt at providing an algorithmic account of learning under this formalism.

[4]And at this point, superficial.

sample of language. The sorts of phenomena the two approaches attempt to explain, as well as the motivations for choosing these phenomena, are also different. The Principles and Parameters researchers search for phenomena which can be cast as universals, while we search for phenomena which we think may be learnable by analyzing a corpus.

## 1.3 Structural Annotation: What is it Good For?

Structural annotation is useful in computational linguistics for a number of reasons, including: extracting meaning from a sentence, checking the well-formedness of a sentence, language modelling and annotating corpora that can then be used as research tools. We will briefly discuss each of these applications in turn.

### 1.3.1 Extracting Meaning From a Sentence

In [82], Marcus argues that it would not be possible to extract meaning from a sentence in general without first obtaining syntactic information. The alternate approach is to assume that the meaning of a sentence can be obtained without recourse to syntactic structure. However, there are many problems with this approach. The first example he gives is the sentence: "The postman bit the dog." If an interpretation of this sentence were to be found based on word meaning and world knowledge, then the sentence would most likely be interpreted so that it is the dog who is doing the biting. However, by knowing that in a simple sentence the noun phrase encoding the actor appears before the verb, we can get the correct interpretation. A simple semantic template-matching approach would fail on many complex sentences, where no keyword matching can uncover the relationships between the words in the sentence. On the other hand, the relationships can be discovered from a structural analysis of the sentence. Even template matching augmented with positional information would be inadequate, since phrases can move out of their base forms. After examining the many pitfalls of any system which tries to extract the meaning of a sentence without referring to structural descriptions, Marcus states: "The purpose of the process of understanding human language is to determine the meanings of utterances, but syntactic structures appear to be a necessary stop along the way."

Theories of compositional semantics such as that proposed by Montague [108] are based upon the assumption that semantic rules are tied to syntactic rules. Therefore, uncovering the syntactic structure of a sentence is a necessary precursor to understanding a sentence in these semantic theories.

### 1.3.2 Sentence Well-Formedness

Structural descriptions of sentences allow for better well-formedness checking than can be done on an unannotated string of words. Take for example, the three sentences:

- (John and Mary) are there.
- ( I called John ) and ( Mary is there).
- I called and ( ( John and Mary ) are there ) .

Without structural annotation, checking the subject/verb agreement for the verb *be* is difficult. In all three sentences, the *be* verb is preceded by the string of words *John and Mary*. With the skeletal bracketing provided in these examples, agreement can easily be checked.

Likewise, information about bracketing and phrasal heads is necessary to check for semantic constraints imposed by the matrix verb on its subject. For example, in the sentences below, we know that ice cream can melt, whereas opera singers cannot. To enforce the semantic constraint of what can be the subject of the verb *melt* in these sentences, we must know the skeletal bracketing shown below, as well as the head of the subject.

- ( The **ice cream** being eaten by the opera singer) melted.
- ?? We ate the ice cream and then watched as (( the opera singer ) melted).

Structural annotation allows for a more complete well-formedness check on a sentence. Checking well-formedness is useful in a number of applications. To give one example, in some speech recognition systems (e.g. [103]), the recognizer outputs a list of the n-best guesses that is then passed to a parser to filter out those sentences that are not well-formed.

The more accurately well-formedness can be assessed, the better these systems that rely on filtering out bad sentences from n-best lists can perform. The same is true for spelling checkers and any other system that outputs a set of possible answers when only one is permitted, where a filter can be used to eliminate certain proposed answers on syntactic grounds. A system with a probabilistic model could just output the sentence with the highest probability, but filtering allows for the system to take certain structural relations into consideration that are not built into the probabilistic model.

### 1.3.3 Language Modelling

Language modelling involves assigning a probability to a string of words. Language models can be used in real-time speech recognition to predict the next word of a stream of language, based upon what last appeared in the stream (e.g. [64]) or to rank alternate theories of what was uttered for filtering or for outputting the most probable theory. Most successful language models are n-gram models, basing the probability of a word on the probability of the preceding n words, or classes of these words (e.g. [65]). Language models based on context free grammars (e.g. [88]) and decision trees (e.g. [2]) have also been proposed. If a speech understanding system translates an utterance as:

The singer sang a lot of a??as

and the sound to language system cannot decide if the final word of the sentence is *arias* or *areas*, a language model could help by indicating that *aria* would be much more likely in this context. A bigram model trained on text other than Opera News, would most likely indicate that *areas* is more likely to follow *of* than *arias* is. A 5-gram model would be needed to capture the relationship between *sang* and *arias*. However, if we have a structural model, we may be able to recognize that *arias* is much more likely than *areas* to be the head of the object of the verb *sang*. One would hope that eventually, a language model based upon the structural description of the language stream would provide a more powerful framework than one based solely upon an unannotated string of words. Of course, it is not necessarily the case that structure will aid in next word prediction, but a cheap source of structural annotation would at least allow this avenue of research to be explored more fully.

### 1.3.4 Corpus Annotation

There has been a growing desire for annotated corpora lately by researchers addressing different issues in linguistics and computational linguistics. Linguists are using structurally annotated corpora to study a number of linguistic phenomena. Hardt [51] uses tagged corpora for a study on VP ellipsis. Niv [87] uses a syntactically annotated corpus to develop a theory about how humans resolve syntactic ambiguity in parsing. Taylor and Kroch [107] use tagged and bracketed corpora of Middle English and Old English for studying diachronic linguistic phenomena.

In computational linguistics, many researchers have been using annotated corpora to train stochastic part of speech taggers and parsers (e.g. [36, 102]). Structurally annotated corpora are being used as the gold standard by which different parsers can be objectively compared [5]. Currently, researchers are limited by the existing annotated corpora and the structural descriptions provided in those corpora, or by sentences that can be successfully annotated by existing taggers and parsers. A system that could automatically annotate a corpus in any language with little human labor required would greatly enhance progress that could be made by researchers using corpora in their work. Even if an adequate annotation accuracy level cannot be obtained using automated procedures, an automated annotator could still be used to bootstrap the process of manually annotating a corpus. In [83], it is shown that manually correcting the output of an automated tagger results in greater speed and accuracy than manually annotating from scratch.

A number of researchers in corpus-based computational linguistics believe that the size of available annotated corpora is the current limiting factor in creating accurate corpus-trained natural language processing systems. If this is the case, the cycle of automatically annotating a corpus, manually correcting it and retraining the automatic annotator on the larger corpus could provide a fast mechanism for providing very large annotated corpora.

## 1.4 Toward Robust and Portable Natural Language Processing Systems

It seems to be very difficult, if not impossible, to manually encode all of the information about a language necessary to make a robust system capable of automatically annotating text with a structural description. For such a system to be effective, a great deal of morphological, lexical, and syntactic information must be made available. In large part due to the highly idiosyncratic behavior of language, manually creating these sources of information is a very difficult task. When providing structural information to the system, one must (at least implicitly) specify the grammar – symbol names and meanings, and the set of allowable rules and relations – by which the information will be encoded. For instance, if it is decided that the grammar will be context-free, a decision has to be made as to the type of nonterminals that will be used (syntactic, semantic, or some combination of the two), the level of specificity of categories, and the actual categories that will be used. This descriptive language and resulting grammar will most likely be language specific, and may even be domain specific. If in the process of encoding linguistic information it becomes clear that the descriptive language of the grammar is not adequate, then substantial recoding to convert the information into a form consistent with the new grammar type must be carried out.

In addition to settling upon an adequate descriptive language for the grammar, one is faced with the problem of writing grammar rules for the linguistic knowledge module. Typically, there are two sources of inspiration for discovering pieces of knowledge that need to be encoded: introspection and trial-and-error. Introspection involves thinking about the facts and phenomena we have learned to be important from our linguistic training and manually recording this information in a way that will make it available for a computer to use in parsing. Trial-and-error involves finding a sentence on which the system fails to work, and adding sufficient information to allow for the processing of this sentence. Both of these methods have their shortcomings. In addition to being labor-intensive, they are complicated by the interaction of various linguistic knowledge modules, and by the interaction of different facts within a single module. Because of the large amount of information and the interactions between various facts, expanding the knowledge base is a tricky endeavor. If the goal of the system is to provide the set of facts and method for combining information that allows for the greatest coverage of the target corpus, then it is by no means clear that the methods of introspection and trial-and-error will converge upon such a grammar. The lack of success to this date in building a robust parser (see [8]) is an indication that perhaps these methods never will.

A system that automatically extracts linguistic generalizations from an annotated corpus has two strong advantages over introspection and trial-and-error. First, automating the development of the knowledge base could greatly reduce the total development time of a system. Second, the statistical property of the learner allows the learner to better quantify the import of different linguistic facts and to weigh different facts in a principled way which is driven by the goal of high coverage, and not biased by linguistic training or the order of sentences on which the system fails. A system based upon the analysis of a corpus can uncover generalizations and weigh the import of different phenomena that are indicated by large data analysis but may not be apparent to a person attempting to hand-code a grammar.

The most successful parsers have been those written for a specific constrained domain, usually including a great deal of domain-specific information. In addition to being difficult to create manually, the resulting language processing systems are expensive to port to new languages or even to new domains. A trainable system would allow for inexpensive porting to new domains that may consist of a completely different grammar specification and set of rules.

It may be that the only viable method for providing a system with the necessary knowledge of language is to have the system learn this information itself by analyzing annotated and unannotated sample corpora. Some degree of automatic training seems a necessity for building robust, portable systems. To what degree systems trained on a corpus can succeed remains to be seen. We hope this work will shed some light on this question.

## 1.5 The Process of Automated Language Learning

We are also interested in studying the learning process itself. If it is indeed the case that systems that learn are the solution to building robust natural language processing systems, then the process of automated language learning deserves further study. There are many different ways one could try to construct a language learner. In [65], a self-organizing language learner is proposed to be used for language modelling. In [6], a method of combining a large manually constructed grammar with statistical information obtained from a large corpus is discussed. In this work we take a different approach, namely starting with a small structurally annotated corpus and a larger unannotated corpus, and using these corpora to learn an ordered list of transformations that can be used to accurately annotate fresh text. By undertaking this work, we can learn to what extent this approach is viable and how this approach compares to other approaches currently being examined.

Figure 1.2 lays out the general framework for corpus-based learning, under which this research is being carried out. The learning system described here begins in a language-naive start state. From the start state it is given an annotated corpus of text as input and arrives at an end state. In this work, the end state is an ordered list of transformations for each particular learning module. The learner is defined by the set of allowable transformations, the scoring function used for learning and the search method carried out in learning. Currently, greedy search is used in all learning modules. At each stage of learning, the learner finds the transformation whose application to the corpus results in the best scoring corpus. Learning proceeds on the corpus that results from applying the learned transformation. This continues until no more transformations can be found whose application results in an improvement (see figure 4.2). Once an ordered list of transformations has been learned, new text is annotated by simply applying each transformation, in order, to the entire corpus (see figure 4.3).

There are a number of interesting properties of this framework which are worth keeping in mind when comparing this approach to other approaches to language learning:

- There is very little linguistic knowledge, and no language-specific knowledge built into the system.

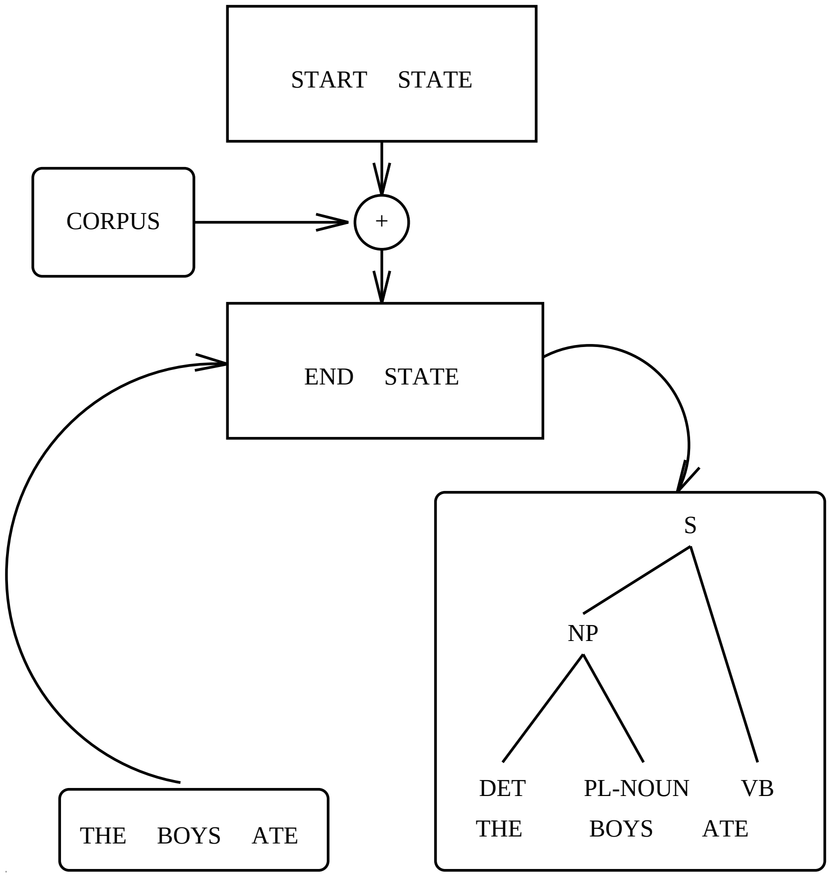



Figure 1.2: General Framework For Corpus-Based Learning.

- Learning is statistical, but only *weakly* so.
- The end state is completely symbolic.
- A small annotated corpus is necessary for learning to succeed.

The learning modules currently implemented are:

- Learning the most likely part of speech for a word.
- Learning how to use contextual information to disambiguate words with more than one part of speech.
- Learning bracketing structure of sentences.
- Learning how to assign nonterminal labels to the bracketing structure.
- Learning how to improve prepositional phrase attachment.

In particular, we will describe a single simple learning method, transformation-based error-driven learning, that has been used to create:

- A syntactic text bracketer that outperforms the best-known statistical grammar induction method, the inside-outside algorithm.
- A part of speech tagger that outperforms statistical taggers based on Markov models.
- A prepositional phrase attachment program that outperforms statistical methods that use t-score statistics.
- A nonterminal node labeller that performs very well despite the fact that very little information is used in labelling.

There are three variables in this system: the level of specificity of the start state, the types of transformation templates, and the degree of annotation of the input corpus. We will assume minimal assumptions about all of these variables, namely a start state with very little linguistic knowledge, very simple language-general transformations, and a small annotated corpus.[5] The less that is prespecified, the easier it will be to port to a corpus from a different domain or in a different language. Also, this way the results obtained will be a lower bound on performance that may be enhanced with larger corpora or more built-in knowledge of language.

This general framework allows for future experimentation with the variables to study how various adjustments affect learning as well as better delineating what pieces of structural knowledge of language can be learned within this framework. Two possible directions worth exploring in the future are annotating the input corpus with varying degrees and types of phrase boundary information and beginning with various linguistic assumptions such as X-bar theory prespecified in the start state.

One possible problem with this line of research arises from our lack of understanding of what is the true structure of a sentence, or even if one *correct* structure exists. Even the structural description of the simple phrase *the boy* is in question. It is unclear whether this phrase is a projection of the noun *boy*, making it a noun phrase, or the projection of the determiner *the*, making it a determiner phrase [1]. With no clear picture of the correct structure of sentences, how can we hope to make progress toward a system capable of learning the information necessary to assign structural descriptions?

This question has to be answered in the context of the current state of the art of language processing. In reality, we are far from the ambitious goal of creating a system that accurately and automatically provides an extremely rich structural description of arbitrary sentences. Given the current level of sophistication of state of the art sentence processors, it is unlikely that progress will currently be hampered by our lack of a detailed understanding of the structure of sentences. As reported in [8], an experiment was recently run in which four large-coverage parsers were presented a number of sentences, all containing fewer than fourteen words. A very generous definition of correctness was used: for a parse to be correct, all that was needed was "accuracy in delimiting and identifying obvious constituents such as noun phrases, prepositional phrases, and clauses, along with at least rough correctness in assigning part-of-speech labels, e.g. a noun could not be labelled as a

[5] Using only an unannotated corpus would have been an even weaker initial assumption, but for reasons explained later we decided to provide the learner with a small annotated corpus which can better be used to guide learning.

verb." One of the parsers scored 60% correct, and the others all scored below 40%. Given the great room for improvement even at this basic level of structure, there is little need currently to be worried if researchers cannot agree upon the proper analysis of complex constructs.

Therefore, progress can currently be evaluated by comparing the performance of a system to the *correct* performance, where correctness can be defined by a manually annotated corpus of skeletal structure. It is possible that at some time in the future, progress will halt. As the problem of *crude* annotation comes closer to being solved and researchers turn toward more elaborate structural annotation, there are two possible pitfalls. One, it may be the case that approaches such as those described in this dissertation are not adequate for uncovering and expressing the more subtle facets of structure. Two, if progress with *crude* annotation outpaces progress in understanding these subtle facets, it will be difficult to create properly annotated corpora which can be used to train the learner and judge progress.

This thesis is an exploration into the power of simple corpus-based analysis as a tool to language discovery. Since this work in many ways parallels the work of Zellig Harris and others from the American Structuralist school of linguistics, we will now review some past work on distributional analysis and automated discovery of structural facts about language.

# Chapter 2

# Structural Linguistics

Although the prominence of structural linguistics has been usurped by modern generative syntax, the goals of the structuralists parallel many of the goals of modern computational linguistics. Both research communities have the structural description of a language as one of the goals of their labor, although the motivations behind this goal are very different. Because of this relationship, we will now briefly examine some past work done in structural linguistics.

According to Sampson [99], Franz Boas is the father of linguistic structuralism. Boas was interested in determining the structure of a number of different languages [11, 62]. Providing an accurate description of each language was the primary goal of this work. From this work, Boas thought, research could be done to determine the relationship between languages based upon their structural similarity. Also, Boas held a view similar to Whorf [112] that the structure of a language influences a person's behavior and therefore saw language study as being important because "the peculiar characteristics of languages are clearly reflected in the views and customs of the peoples of the world" ([62], page 69).

This view, which gave import to studying the structure of individual languages in isolation, is a fundamentally different focus from modern syntacticians, who study a particular language in hopes of learning more about human language in general. Boas believed that since human languages were richly and arbitrarily diverse, approaching the problem of describing a language with preconceived linguistic notions would not be fruitful, and could even lead to wrong analyses. Also, since linguistic facts do not easily rise into consciousness, a method of analysis is necessary to elicit these facts. He proposed using a form of distributional analysis. As an example, a linguist analyzing English could determine that *m* and *n* are not allophones by noting that the sounds *mail* and *nail* convey different meaning.

Boas' work was followed by that of Leonard Bloomfield [9]. Bloomfield also worked on uncovering descriptions of unfamiliar languages. And like Boas, Bloomfield believed that when studying an unfamiliar language, one had to be extremely careful not to allow any preconceived notions to creep into the study. Bloomfield says (page 20):

> The only useful generalizations about language are inductive generalizations. Features which we think ought to be universal may be absent from the very next language that becomes accessible. Some features, such as, for instance, the distinction of verb-like and noun-like words as separate parts of speech, are common to many languages, but lacking in others. The fact that some features are, at any rate, widespread, is worthy of notice and calls for an explanation; when we have adequate data about many languages, we shall have to return to the problem of general grammar and to explain these similarities and divergences, but this study, when it comes, wil be not speculative but inductive.

Bloomfield was heavily influenced by logical positivism ([99]). In logical positivism, there was no room for theories based upon anything but simple, indisputable sensory data. Therefore, Bloomfield believed that a linguist could not use introspection as a way of gathering linguistic data, but could only rely upon actual utterances gathered in field work. Bloomfield elaborated upon Boas' method of using distributional information from a corpus of actual utterances to draw conclusions about a language. By studying the behavior of elements in the corpus, one can draw conclusions about the forms and grammar of an unfamiliar language. One thing lacking in Bloomfield's work is a formal, algorithmic description of how one can extract structural information from a corpus. As computers became more prominent, this weakness became more significant.

Zellig Harris attempted to describe the structuralist idea of language analysis with sufficient rigor so that it could conceivably be written as a computer program. Harris developed rules that a linguist doing field work could use to uncover the structure of an unfamiliar language [53]. In addition to the hope of eventually automating the process, Harris was troubled by the lack of rigor in the analysis linguists carried out on data collected from field work. He hoped that by providing a set of procedures for the linguist to use in his or her analysis, the lack of rigor could be overcome. It is important to emphasize that Harris was not putting forth a theory of grammar, nor was he claiming to have a theory of language learning; rather, he was developing tools that a linguist (or a computer) could use to help build a theory or structural description of a language. Of course, the sorts of things the linguist would be likely to discover are influenced by the tools used, and using the tools outlined by Harris commits one to a particular class of language theories.

The methods Harris proposed were layered to first discover morphemes from phonemes, then word classes from morphemes, and then higher level structure from words and word classes. The methods used were all based upon the observation of the set of environments different elements are found in. Below we describe three different discovery procedures posed by Harris and his contemporaries.

## 2.1 Discovering Morpheme and Word Boundaries

Harris proposed a method to discover morpheme boundaries within a word and word boundaries within a sentence [53, 54]. The procedure is given a sentence as input, transcribed either in phonemes or letters. For each prefix of the sentence, the number of phonemes that can follow is computed. When this procedure is carried out, the number of allowable phonemes gradually decreases as more of a word is included in the prefix. Then, when a word or morpheme boundary is reached, the number of allowable phonemes greatly increases. This is because a morpheme is distributionally much freer than a morpheme prefix, and therefore there is greater variation in what can appear after a morpheme. By computing this value over a sentence in both the forward and backward directions, the peaks correspond to word and morpheme boundaries. The procedure is carried out in both directions to make it more robust.

When Harris ran the algorithm to break words into morphemes, he tried the words *disembody* and *disulfide*. If the algorithm were only run in the forward direction, it could not be distinguished whether *di-* or *dis-* was the proper prefix for these two words. By running the algorithm backwards, the proper decomposition can be found. This is because *sulfide* is distributionally freer with respect to what it can follow than *ulfide*, and *embody* is freer than *sembody*. When Harris ran the algorithm on words, he used a dictionary of English to compute the number of letters that can precede (follow) a prefix (suffix). Presumably, one would not have access to a large dictionary when doing field work on a little-known language. Without a dictionary, one could proceed in two ways. If a sufficiently large corpus could be obtained, a word list could be built from the corpus and the numbers computed from this word list. Or, if an informant was available, the informant could be queried as to the number of possible completions he could think of for a particular prefix. While the method of using a corpus may be possible to find morphemes when word boundaries are already known, it could not be used to find word breaks in a phonemic transcription. As Harris states, the corpus that would be needed for such an analysis would be prohibitively large.

## 2.2 Discovering Word Classes

According to Harris, the motivation for grouping words into classes when building a structural description for a language is to avoid having to repeat identical grammar rules for different, but similar, words. By grouping similar words together, the grammar can be expressed more economically. Since the discovery procedure is layered to learn less complex classes and relationships first, we can presume that we discover the morphemes of the language being studied first, and that classification can then proceed over these known morphemes. Harris proposes that two words that can occur in the same environments can be classed together, where an environment is simply a context in which the word appears. For instance, the word *boy* can appear in the environment *The fastest ___ won the race.* Since very few word pairs are completely identical with regard to the set of environments they can appear in, the constraint can be weakened to allow two words to be classed together if a sufficiently large percentage of environments are shared. At the end of this classification procedure, classes will be found such that the set of allowable environments of every word in the class is roughly the same, and there is a significant distributional distinction between any two words of different classes.

The word classification procedure was not completely automatic. Since it would be impossible to obtain a corpus of utterances sufficiently large to contain most environments that each word to be classed can appear in, approximation techniques were employed. One possible approximation technique is for the linguist to search for short environments that are good at differentiating classes, such as a small set of suffixes. For the linguist to successfully identify good diagnostic environments, he must be familiar with the language being studied, or have access to an informant.

## 2.3 Discovering Significant Morpheme Strings

In the work of the American Structuralists, we find a number of suggestions as to how an immediate constituent analysis can be performed on a sentence.[1] Seymour Chatman [33] and Charles Hockett [61] have suggested using a measure somewhat similar to entropy as a tool for breaking a sentence into phrases. Chatman proposes that "the greater the potential variety of following environment (that is, the greater the number of possible morpheme substitution classes which can immediately follow a string of morphemes), the greater the magnitude of the structural break which separates the morpheme from what follows." To determine the strongest break in the sentence *the hungry boy ate*, one would query an informant to determine the number of different word classes that can follow *the ...*, *the hungry ...*, and *the hungry boy ...*. Note that this is different from an entropy measure, which would take into account the probabilities of word classes appearing in each context. In structural linguistics, an informant provides a binary answer indicating whether or not an entity can appear in a particular environment. When extracting information from a corpus, we have an estimate of the probability of an entity appearing in that environment. This method of finding phrase boundaries is similar to that proposed by Harris [54] for determining the morphemes of a language.

A different approach to immediate constituent analysis has been suggested by Zellig Harris [52, 53] and Rulon Wells [111]. Since the work of Wells incorporates and expands upon the ideas of Harris, we will only discuss Wells' work here. It is possible for two

[1] But not automatically.

different word class sequences to be substitutable for each other; in other words, in every sentence that one sequence can occur, the other can occur as well. The example Wells gives is that the word class sequence for *Tom and Dick* and that for *they* are mutually substitutable. Given two word class sequences $A$ and $B$, Wells calls $A$ an expansion of $B$ if $A$ and $B$ are mutually substitutable, $A$ and $B$ are structurally diverse[2], and $A$ contains at least as many morphemes as $B$. A word class sequence $A$ is said to be an expansion if there exists some $B$ which it is an expansion of. Immediate constituent analysis is carried out by attempting to break a sentence into word sequences that are expansions. To get around the problem of not having access to all distributional possibilities in a corpus, Harris suggests that the linguist construct testing frames for each word class. Testing frames are environments that the linguist deems representative of a particular class. Once these testing frames are chosen, word sequences can then be found that can naturally appear in all of the testing frames for a particular word class. Using Harris' example, this procedure would equate the sequence *adjective noun* with the word class *noun*, since for instance both *good boy* and *fool* can appear in the testing frame *Don't be a* ___.

Of these three procedures, only the discovery procedure for morpheme and word boundaries was developed to the point where it could be implemented and tested on a computer. The other procedures relied upon the intelligence and active intervention of a linguist to decide the best questions to ask an informant or to decide what specific environments should be searched for in a corpus of utterances. For instance, in his discussion of word classes, he first suggests determining similarity by looking for words that have identical distributional behavior in short environments, instead of looking for words that can appear in precisely the same sentences. But then Harris notes a weakness with this approach.

> This method, however, may not prove adequate. In many languages it may be impossible to devise a procedure for determining which short environments over what limits should be set up as the differentiating ones for various substitution classes. If we select *-ing* as a diagnostic environment, we would get a class containing *do, have, see, etc.,* but not *certain*. If we select *un-* as the environment, we obtain a class with *do, certain, etc.*, but not *have*, and with

[2] It is not clear precisely what is meant by structural diversity.

> *see* only if *-en* or *-ing* follow. We could obtain many different classifications of the same morphemes.

In the work described in this dissertation, we address learning some of the structural information about language that the structural linguists developed procedures to elicit from informants. In our work, once an informant has annotated a small, randomly selected sample of language, all learning is automatic. Whereas the field linguist working with an informant in essence had access to the intensional distribution of the language being discovered, we make use of an extensional distribution observed in a small naturally occurring sample of annotated text. We have also expanded the idea of distributional analysis in a novel way: instead of examining the distribution of entities in a corpus, a naive first guess is made as to the structure of the language, and then an analysis of the distribution of errors is carried out to discover transformations to eliminate annotation errors.

The distributional hypothesis states that lexical features and syntactic phenomena manifest themselves in a way that can be observed in surface-apparent distributional behavior. If this hypothesis is false, then techniques of the sort outlined above and in this thesis will never be completely successful. Since the extent to which the distributional hypothesis holds can only be judged through the success or failure of approaches based upon the hypothesis, it must be tested empirically just how far distributional techniques can go.

# Chapter 3

# Some Recent Work on Corpus-Based Learning

With the advent of large on-line corpora and fast computers, there has been a great deal of excitement over the last few years in trying to automatically extract linguistic knowledge from text corpora. This movement is in essence a rebirth of structuralism, appropriately adapted to the age of fast computers and cheap storage devices. To what extent such algorithms can succeed at extracting useful information is an empirical question. Issues such as whether distributional information is sufficient, what size corpus is needed to access the information, what knowledge of language needs to be built into the learner, and whether the noise in the corpus is harmless cannot be solved through intuition; only experimentation will answer these questions. Over the last few years a number of surprising successes have demonstrated the strength of these methods, as well as demonstrating some weaknesses inherent in the approach. We will review a few of these results below.

## 3.1 Annotating a Corpus With Part of Speech Labels

The need for annotated corpora has grown over the past few years. A number of corpora with words tagged for part of speech are now readily available and are heavily used by natural language researchers. Tools to automatically tag a text with parts of speech have been very successful. Using these tools to tag text and then having people correct mistakes manually has resulted in very fast and accurate tagging of large amounts of text [22, 83]. Although a bit circular, building larger corpora provides training material to build more accurate automatic annotators which can then be used in applications that require annotated input, or to build even larger annotated corpora.

Part of speech tagging involves assigning every word its proper part of speech based upon the context the word appears in. For instance, in the sentence below, *can* has three different part of speech tags.

Can/MODAL we/PRONOUN can/VERB the/DETERMINER can/NOUN ?/?

There are two pieces of information needed for tagging. Lexical information indicates the possible parts of speech for particular words, possibly including some indication of likelihoods of different labels. Contextual information indicates the particular tag that is appropriate for a particular context.

A number of systems have been built which are quite effective at accurately tagging text. Until recently, the most effective have been statistical, Markov-model based taggers.[1] There are two general classes of statistical taggers: those trained on tagged text, and those trained on untagged text. The underlying model is a set of part of speech states, with each state generating different words with different probabilities. For instance, to generate the sentence *the dog barked*, the model would begin in a *determiner* state, from which the word *the* would be emitted, then move to a *noun* state, from which the word *dog* would be emitted, and then finally move to a *verb* state which would emit the word *barked*. Given a string of words, the goal is to uncover the sequence of states that generated the string. When a tagger is trained on tagged text, the state transitions are visible, and so the transition probabilities and the emit probabilities are easy to estimate from the training corpus. The taggers described in [36, 40, 45, 84] are trained on tagged text. From a large corpus of tagged text, a set of lexical and contextual probabilities are estimated. Lexical probabilities are $P(W|T)$, the probability of a word given a part of speech tag. In other words, $P(eat|verb)$ is the probability that if a word is labelled as a verb in the corpus, then the word will be *eat*. Contextual probabilities are computed as $P(T_i|T_{i-1})$ or $P(T_i|T_{i-1}T_{i-2})$, depending upon the size of the context window being used. Once the

[1]For a good introduction to Markov models, see [93].

system is trained, new text can be tagged by assigning the string of tags which maximizes $P(W|T) * P(T_i|T_{i-1})$ for a sentence. This optimal tagging can easily be computed using dynamic programming [109].

The second set of stochastic taggers does not require tagged text for training. The same underlying model is assumed, namely a Hidden Markov model, but in this case training is more difficult because the set of state transitions used to generate the training corpus is no longer visible. Although a tagged corpus is not necessary, a large dictionary is necessary to determine the permissible part of speech tags for words. If an on-line dictionary is not available for the language of the corpus being tagged, or if the tags in the dictionary cannot be mapped into the desired set of tags, then a great deal of human labor is required to provide this necessary training material. The taggers described in [39, 65, 73] are of this type. They use the Baum-Welch algorithm [4] to train the model, and then use this trained model for tagging fresh text. It is not clear whether this approach of training on untagged text provides an effective and portable method of tagging. For example, in [73], performance comparable to that obtained by taggers trained on tagged corpora is obtained. However, to obtain this performance, a large dictionary with part of speech and inflectional information was needed, and a number of higher-order procedures were manually built based on manual error analysis. In addition, the results quoted are based on lexical information obtained from both the training and test set, and it is not yet clear if this can obtain accuracy comparable to taggers trained on tagged text.

There have been a number of attempts at rule-based tagging as well. Rule-based taggers date back as far as [55, 71], but only recently, with the availability of fast computers and large corpora, have these taggers been able to tag with extremely high accuracy. In [57], part of speech tagging rules are discovered automatically within a sophisticated Marcus-style parser [81]. Rules make reference to the state of the parser during the processing of the word being tagged. In [16], a simple rule-based tagger is described. In [7], a decision tree is used in tagging.

In all of these approaches, contextual information is used to disambiguate from an already known set of allowable part of speech tags. A problem arises when a word is encountered for which no part of speech information is known. All statistical taggers must deal with smoothing in some way, since an empirical probability estimate of zero can often lead to errors. One problem with current successful approaches to tagging is that none of them handle unknown words in a way that is completely portable. In [36], Church hard-codes a complex procedure for classifying unknown words, including a frequency-dependent procedure for detecting proper nouns, a domain-dependent procedure for classifying words with dashes, a list of abbreviations, a large list of informative suffixes, and a great deal of additional information. In [73], Kupiec provides a list of closed class tags, and assumes that the external dictionary will always list all closed class items. In addition, he provides a set of derivational and inflectional suffixes and then trains a probabilistic method for determining their part of speech, trained on the dictionary and a corpus of unannotated text. In [84], a probabilistic procedure is also employed for unknown words. This procedure also requires that an informative set of affixes be provided, as well as other likely cues. In a later chapter, we will discuss a transformation-based learner that automatically learns to tag unfamiliar words with no prior language-specific knowledge.

It is interesting to note that all of these taggers obtain roughly the same performance when trained and tested on comparable corpora, when controlling for such variables as the size of the dictionary used and the amount of morphological information used. These variables can significantly effect performance, and without factoring this in, we cannot be sure if we are measuring the success of a tagging method, or merely the success of the extra information provided. In light of the comparable performance achieved by all taggers, that described in [16] and below is much simpler than the others. For example, contextual information is captured in fewer than 100 rules in [16], compared to a 30,000 to 40,000 leaf decision tree in [7] and a table of tens of thousands of contextual probabilities in [36].

## 3.2 Learning Lexical Information

Distributional techniques have also been useful in helping a lexicographer uncover lexical information about words that he might not have been able to think of through introspection.

Recently developed techniques use mutual information, a measure of how the cooccurrence of two elements compares with chance. The mutual information of two events x and y is defined as:

$$I(x,y) = log_2 \frac{P(x,y)}{P(x) * P(y)}$$

If x and y appear together only by chance, then $I(x,y) = 0$; If $I(x,y) > 0$, then they occur together more than chance would predict. If $I(x,y) < 0$, then they occur together less than chance would predict. One would expect I(clouds,rain) to be positive, I(rain, sunshine) to be negative, and I(even numbered day, rain) to be zero.

In [37] it is shown how one can use the mutual information statistic to uncover lexical information. In this paper, they compute the mutual information of *strong* ___ and *powerful* ___ for all words that occur next to these two words in the 1988 Associated Press newswire. The list of five highest scoring neighbors for both *strong* and *powerful* is shown below.

| I(x,y) | x | y |
|---|---|---|
| 10.5 | strong | northerly |
| 9.8 | strong | showings |
| 9.3 | strong | believer |
| 9.2 | strong | second-place |
| 9.2 | strong | runup |
| 8.7 | powerful | legacy |
| 8.6 | powerful | tool |
| 8.4 | powerful | storms |
| 8.3 | powerful | minority |
| 8.1 | powerful | neighbor |

A similar list can be computed on word pairs that have relationships other than immediate neighbor. For instance, in [58], nouns are classified based on the mutual information between them and verbs they are the argument of. From a list such as this, a lexicographer could uncover subtle differences between words that he may not have thought of without the aid of such a list.

In [47], a semi-automated procedure is described for learning what classes of objects in a subdomain can enter into a subject-verb-object relationship. However, the procedure needs a reliable parser, and a great deal of human intervention. [14] describes a procedure for extracting verb subcategorization information from an unannotated text. The verb subcategorization frames that it finds include: direct object, direct object and clause, direct object and infinitive, clause, and infinitive. This procedure works without a parser and, once written, needs no human supervision. First, a list of verbs is extracted from the corpus. This is done using a simple automaton that assumes a word is a verb "if it is adjacent to a pronoun or proper name that would otherwise lack case." The verb finding algorithm is based upon the Case Filter [98], which states that for a noun phrase to get case in English, it must occur in one of a small number of possible positions in a sentence: immediately to the left of a tensed verb, to the right of a main verb, or to the right of a preposition. The system is given a list of prepositions, so if it can recognize noun phrases, it can determine where the noun phrase must get case from a verb, and thereby can detect the verbs in the corpus. Since noun phrases are not trivial to detect, only pronouns and proper names, which are easily detected noun phrases, are considered. Automata are then built manually to detect a number of different subcategorization frames based only upon the easily detected and unambiguous instances of those frames. This method has proved to be effective at extracting verbs and detecting, from the prespecified set of subcategorization frames, those that each verb can appear in. The accuracy in detecting instances of each of the five subcategorization frames mentioned above ranged from 97% to 99.5%. However, to apply this technique to a new language, a new verb extraction program would have to be written, as well as new automata to recognize subcategorization frames.

## 3.3 Learning Phrase Structure

Automatically learning information that can be used to accurately assign a phrase structure analysis to sentences has been the topic of a number of recent papers. A number of papers from the school of structural linguistics addressed this issue. In [106], using mutual information (called *interword predictability* by Stolz) to discover phrases is suggested, with the crucial insight being that local minima in interword predictability correlate well with phrase boundaries. In [79], this idea is elaborated upon and tested on a large corpus.

In [49, 100], simulated annealing is used to parse a sentence. First, a scoring function is defined that can take any tree structure as input and score the quality of that tree. Then a set of moves is defined, which includes changing the nonterminal label of a node and restructuring a tree. Parsing is then carried out using simulated annealing to move through the search space in hope of ending up with a high scoring tree.

In [23], distributional analysis techniques similar to those described in [111] are used to automatically learn scored context-free rules. The score for the rule:

$$noun \rightarrow determiner\ noun$$

receives as its score the distributional similarity, based on words immediately to the left and right, of the single part of speech *noun* to the part of speech pair *determiner noun*. Parsing is carried out by repeatedly reducing a pair of tags to a single tag in a way that maximizes the similarity of the two items involved in every reduction.

In [12], statistics are calculated on all possible subtrees contained in a structurally annotated training corpus. Since finding the optimal, or highest probability, combination of subtrees would result in a parser requiring exponential time, a Monte Carlo technique [50] is used to find a good guess at the optimal combination of subtrees when parsing fresh text.

The inside-outside algorithm [3] is a method for training stochastic context-free grammars. It is an extension of the Baum-Welch [4] algorithm for training stochastic finite state automata.[2] A number of recent papers have explored the potential of using the i-o algorithm to automatically learn a grammar [75, 104, 88, 26, 31, 102]. A probabilistic context free grammar begins with some initial, possibly random, probabilities. The inside-outside algorithm is an estimation maximization algorithm which iteratively changes the rule probabilities to increase the probability of the training corpus. The algorithm is guaranteed to find a locally optimal assignment of rule probabilities, but not a globally optimal assignment. In [88, 102], it is shown that the inside-outside algorithm can be used to bracket text with high accuracy, with very weak initial knowledge of the grammar. In [8], the inside-outside algorithm is used to convert a grammar written by a linguist into a probabilistic grammar where the hope is that the most probable parse is often the *correct* parse.

[2] For a good tutorial on the inside-outside algorithm, see [66].

In this thesis, we will discuss an error-driven approach to learning a grammar for bracketing text. The approach works by beginning with a very naive parser, and then learning a set of transformations that can be applied to the output of the parser to make parsing more accurate. We will show that this method achieves performance comparable to that achieved by the inside-outside algorithm. One interesting thing about the error-driven approach is that unlike almost all other recent attempts at grammar induction, the resulting grammar is purely symbolic and the learning process is only weakly nonsymbolic.

## 3.4 Other Areas

In this chapter, we have only touched upon a few of the many research programs based on extracting various sorts of linguistic information from corpora. Other areas include machine translation [27], word sense disambiguation [28], word clustering [22, 15, 29, 89], and pronoun resolution [30].

# Chapter 4

# Transformation-Based Error-Driven Learning Applied to Natural Language

## 4.1 Introduction

In this section, we describe a framework for learning which has been effectively applied to a number of language learning problems. We call this framework *transformation-based error-driven learning*. In this learning paradigm (see figure 4.1), unannotated text is first presented to the system. The system uses its prespecified initial state knowledge to annotate the text. This initial state can be at any level of sophistication, ranging from an annotator that assigns random structure to a mature hand-crafted annotator. In work described herein, the initial state is never a difficult state of knowledge to obtain: it is always either a trivial algorithm or contains information derived automatically from a corpus. In the module that tries to find the most likely part of speech tag for every unrecognized word, the initial state system assumes that every word is most likely a noun. In the part of speech contextual disambiguation module, the initial state system assigns every word its most likely tag, as estimated from the small annotated training corpus. In the phrase structure bracketing module, the initial state system assigns a right-linear structure with final punctuation attached high to all input sentences. In prepositional phrase attachment, prepositional phrases are always attached low. In nonterminal node labelling, a node is labelled with the most likely tag to dominate its daughters, or a default tag if the string of daughters was not seen in the training corpus. There are two important observations to make about the initial state annotators used in this dissertation. First, it should be clear that they are all extremely simple to create and contain no language-specific knowledge. If any start state knowledge turns out to be language specific, it can easily be parameterized. For instance, left-branching bracketing may prove to be a more effective start state than right-branching bracketing for some languages. The start state bracketer could then be parameterized, and only a small amount of annotated text would be needed to determine the proper parameter setting. This makes the learner highly portable. Second, the initial-state annotators will perform terribly on their own. Rather than manually creating a system with mature linguistic knowledge, the system begins in a naive initial state and then learns linguistic knowledge automatically from a corpus.

After the text is annotated by the initial state annotator, it is then compared to the *true* annotation as indicated by the annotation assigned in the manually annotated training corpus. We have used three different manually annotated corpora: the Penn Treebank [22, 83] and original Brown Corpus [44] for experiments in English and a manually annotated corpus of Old English [20]. Note that the main expense in writing and training the learning programs in this learning paradigm is in creating the small annotated corpus. Fortunately, the learning methods do not require a great amount of annotated text for learning. At most, 45,000 words of annotated text were used in our experiments.[1] This is a small enough corpus that it is not a significant cost in time to have an informant annotate such a training corpus. Future research into more powerful transformations will hopefully allow for comparable performance on an even smaller training corpus. In addition, the process of manually annotating can be sped up by repeatedly annotating a small amount of text, training the automatic annotator on that text, having the automatic annotator annotate some new text, and then manually correcting the output of the automatic annotator. It is much faster to correct annotation errors than to annotate from scratch [83].

[1]This could possibly be cut in half. Currently, the lexical and contextual modules are trained on separate annotated corpora. This is so the behavior of unknown words when training the contextual module will be similar to that of fresh text. If another way of accomplishing this could be used, then the two training corpora could overlap, thereby greatly reducing the total annotated text requirements of the system.

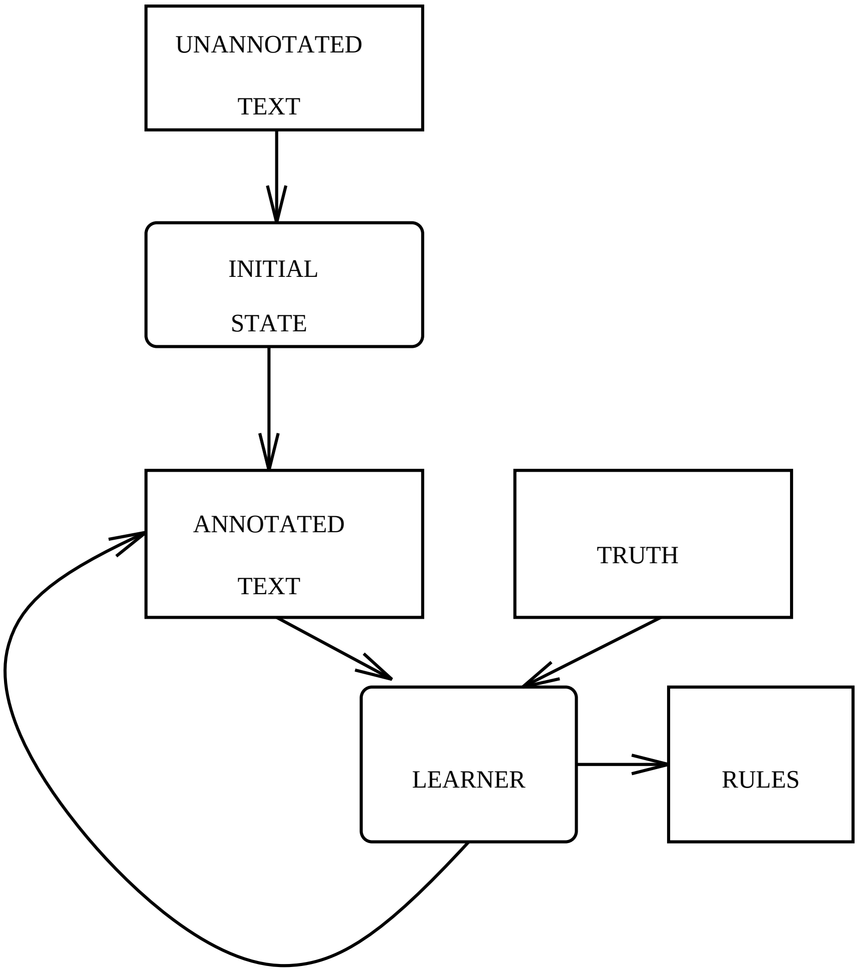



Figure 4.1: Distributional Error-Driven Learning.

By comparing the output of the naive start-state annotator to the *true* annotation indicated in the manually annotated corpus, we can learn something about the errors produced by the naive annotator. Transformations can then be learned which can be applied to the naively annotated text to make it better resemble the manual annotation. A set of transformation templates specifying the types of transformations which can be applied to the corpus is prespecified. In all of the learning modules described in this dissertation, the transformation templates are very simple, and do not contain any deep linguistic knowledge. The number of transformation templates is also small. Transformation templates contain uninstantiated variables. For instance, in the template:

Change a tag from **X** to **Y** if the previous tag is **Z**

X, Y, and Z are variables. All possible instantiations of all specified templates defines the set of allowable transformations.

The application of some transformations will adversely affect the quality of annotation, resulting in further divergence from the manually annotated treebank, while others will result in a more accurately annotated corpus. The learner searches for that transformation whose application will result in the greatest improvement of annotation quality, which can easily be measured by applying the transformation and comparing the resulting annotations to the manually annotated treebank. The best transformation is recorded in the ordered set of learned transformations and is applied to the training corpus. Learning then continues, as the learner trys to find the best transformation for the corpus annotation that results from applying the first learned transformation to the training corpus. Learning stops when no more effective transformations can be found, meaning either no transformations are found that improve performance, or none improve performance above some threshold.

Figure 4.2 outlines the learning process. In this example, the initial corpus has 532 errors, found by comparing the annotated corpus to a gold standard, namely a manually annotated corpus. At time T-0, all possible transformations are tested. Transformation T-0-1 (transformation T1 applied at time 0) is applied to the corpus, resulting in a new corpus: Corpus-1-1. There are 341 errors in this corpus. Transformation T-0-2, obtained by applying transformation T2 to corpus C-0 (which is obtained using the initial-state annotator), results in Corpus-1-2, which has 379 errors. The third transformation results in an annotated corpus with 711 errors. Because Corpus-1-1 has the lowest error rate, the transformation T1 becomes the first learned transformation, and learning continues on Corpus-1-1, the corpus resulting from applying transformation T1 to corpus C-0.

We will show that transformation-based error-driven learning is an effective learning method in a number of structural language learning tasks, including part of speech tagging, prepositional phrase attachment and parsing.

Although any measure of success can be used to guide learning, a very coarse-grained measure will probably not lead to successful learning. For instance, in learning bracketing transformations, we use a measure that is a function of all brackets. This means that a small change will affect the measure. If instead we used a much more coarse grained measure for learning such as the number of bracketed sentences in the training corpus that precisely match the bracketing in the manually annotated corpus, this measure would not be sufficiently sensitive to minor bracketing changes to adequately guide the search.

We have currently explored only one learning method for obtaining an ordered list of transformations: a greedy algorithm at each stage adds the transformation with the highest success score. One could use other control strategies, such as search with a look-ahead of greater than one transformation, or other strategies for dealing with a large search space such as simulated annealing [69] or a genetic algorithm [46].

In transformation-based error-driven learning, there are two pieces of knowledge that need to be prespecified: the start state annotation algorithm and the set of transformation templates. The prespecified knowledge is very cheap to create. Once it is created, there is no cost in porting it to a different domain or language, other than obtaining a small annotated corpus. The start state and transformation templates are completely general; it is the interaction of the learner with the training corpus that results in domain or language specific knowledge being obtained. Once learning is completed, new text can be annotated simply by passing it through the start-state annotator, and then applying each of the learned transformations, in order. In figure 4.3, Corpus-0 is obtained by applying the initial-state annotator. The first transformation is applied to the entire corpus, resulting in Corpus-1. The second transformation is applied to Corpus-1, and so on until the list of transformations is exhausted.

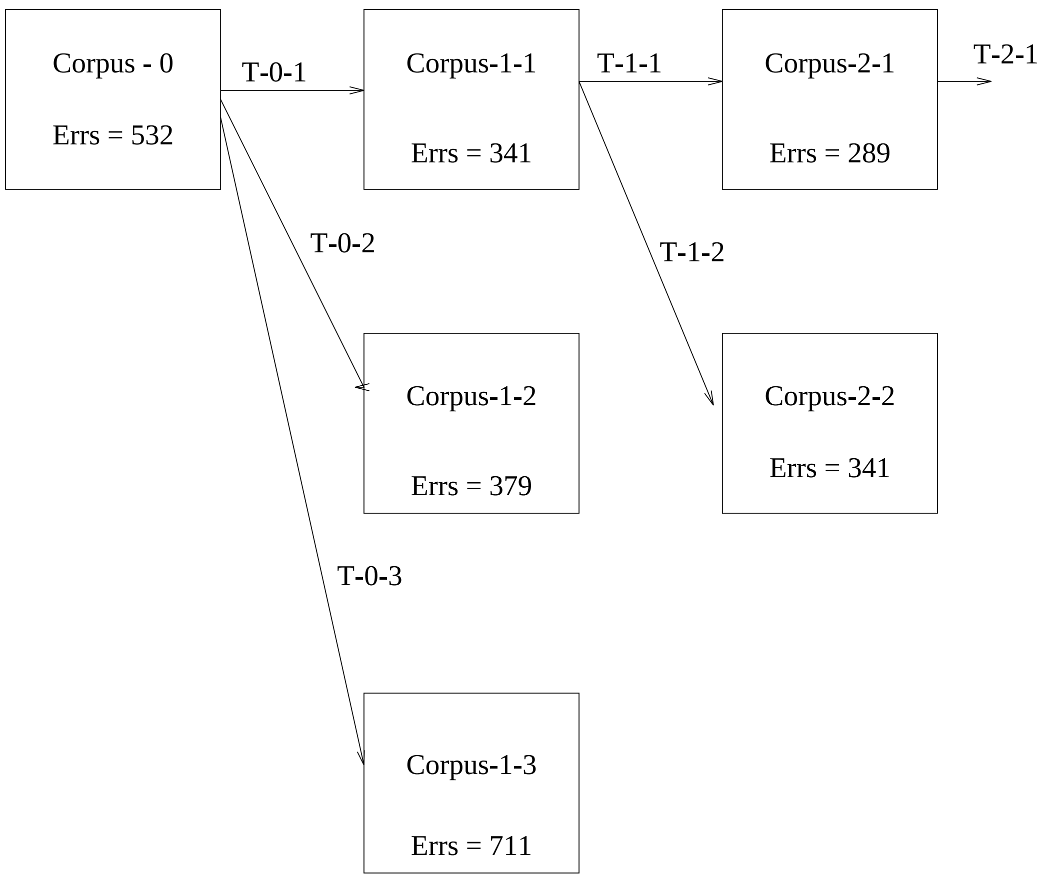



Figure 4.2: Learning Transformations

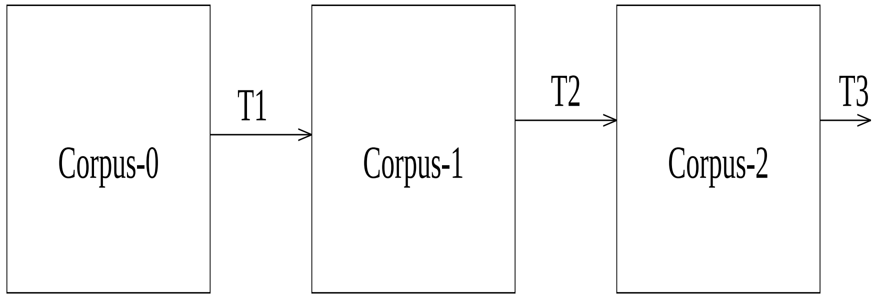



Figure 4.3: Applying Transformations

Transformation-based error-driven learning is a degenerate instance of means-ends analysis. GPS (General Problem Solver) [41, 86] is probably the earliest successful implementation of a means-ends analysis system. In GPS, a set of rules is specified. Rules have two parts: the preconditions that must be satisfied to trigger a rule, and the effect of carrying out the rule. The search strategy employed in GPS is more complex than that of our learner. In GPS, a problem is decomposed into a set of easier problems in a way that will better enable the system to lessen the difference between the current state and the desired end state. The transformation-based learner decomposes the problem of getting from a naive annotation to the proper annotation into a set of subproblems, iteratively taking the biggest improvement step possible. Unlike general means-ends analysis, states are not saved, and backtracking is never employed. Progress is always made in a forward direction from current state to goal, and never backwards from goal to current state. In addition, transformations are learned in the transformation-based learner, whereas the rules of GPS are prespecified.

The technique employed by the learner is also similar to that used in decision trees [13, 91, 92]. A decision tree is trained on a set of preclassified entities and outputs a set of questions that can be asked about an entity to determine its proper classification. The tree is built by finding the attribute whose distribution has the highest entropy in the training set, asking a question about that attribute, splitting the training set according to that attribute value, and then recursively reapplying this procedure on each resulting subset. In natural language, decision trees have been applied to language modelling [2] and part of speech tagging [7]. One crucial difference between training decision trees and training the transformation-based learner is that when training a decision tree, each time the depth of the tree is increased, the average amount of training material available per node at that new depth is halved (for a binary tree). In transformation-based learning, the entire training corpus is used for finding all transformations. In addition, transformations are ordered, with later transformations being dependent upon the outcome of applying earlier transformations. For instance, whether the previous word is tagged as *to-infinitival* or *to-preposition* may be a good cue for determining the part of speech of a word. If initially the word *to* is not reliably tagged everywhere in the corpus with its proper tag, then this cue will be unreliable. The transformation-based learner can delay positing a transformation triggered by the tag of the word *to* until other transformations have resulted in a more reliable tagging of this word in the corpus. The transformation-based learner is also considerably simpler than decision-tree learning, using simpler mathematical techniques, and requiring no smoothing or pruning of trees. In addition, the resulting learned information is much more compact in transformation-based learning. For example, in the application of part of speech tagging, the decision-tree tagger described in [7] outputs a tree with 30,000 to 40,000 leaves, whereas the transformation-based learner outputs a list of fewer than 200 transformations.

A decision list [95] is similar to a decision tree, except that it is restricted to being binary-branching and right-linear. In other words, a decision list is a set of statements of the form:

$$\text{If } X_1 \text{ then } Y_1 \quad \text{else if } X_2 \text{ then } Y_2 \quad \ldots \quad \text{else if } X_n \text{ then } Y_n \quad \text{else Z}$$

where $X_i$ are questions, $Y_i$ are classifications, and Z is a default classification to apply if no questions in the decision list are answered positive. The main difference between a decision list and an ordered set of transformations is that more than one transformation can apply to a single triggering environment.

In transformation-based error-driven learning, once a set of transformations is learned the application order is completely specified and deterministic. This is different from other approaches to annotation. When parsing with a context-free grammar, an algorithm must examine different possible combinations of rules to find the set of rules that together can generate the sentence. In statistical part of speech tagging, dynamic programming is used to find the highest probability path through a set of states. The transformation-based approach assigns a structural annotation to all input sentences, including sentences exhibiting phenomena not observed in the training corpus and noisy input. This is because this approach works by first assigning some default annotation structure to sentences, and then altering that structure based on triggering environments. This is different from parsing with a grammar, where a set of rules must account for the relationship between all tokens in the input and will fail to parse if a sentence is not covered by the grammar.[2]

[2] Proposals for handling sentences not covered by a grammar have been discussed. For example, see [32].

For error-driven learning to succeed, it must be the case that a set of ordered transformations can be learned whose application significantly improves performance over accuracy obtained by simply using start-state information. It must also be the case that a transformation whose application proves fruitful in the learning process will also prove fruitful on text other than the training corpus. For error-driven learning to be computationally feasible, it must be easy to apply the set of operations and to recognize the set of triggering environments. Since the run-time of the learning algorithm is $O(|op| * |env| * |n|)$, where $|op|$ is the number of allowable transformation operations, $|env|$ is the number of possible triggering environments, and $|n|$ is the training corpus size, a large set of operations or environments will make learning computationally infeasible.[3]

In later chapters we will detail how error-driven learning has been successfully applied in a number of domains, including part of speech tagging, prepositional phrase attachment, and parsing. To help solidify the ideas described in this section, we will briefly outline the error-driven part of speech tagger we have developed [16] (which we discuss in more detail in a later chapter). In this system, the initial state is an algorithm that tags every word with its most probable tag in isolation, along with a guessing procedure for unknown words. Allowable operations are of the form *change part of speech tag from X to Y*, for all X,Y in a predefined set of part of speech tags. The set of triggering environments includes:

1. The current word is W.
2. The previous word is tagged as T.
3. The following word is tagged as T.

We have found that this transformation-based tagger, after learning fewer than 100 transformations, obtains tagging accuracy comparable to state of the art stochastic taggers in spite of the fact that the resulting knowledge base is considerably smaller and is entirely nonstatistical.[4]

To understand the success of error-driven learning, we have to examine rank-frequency distributions. If, at a particular stage of learning, the training set only has one instance

[3]As will be explained later, since the learning algorithm is data driven, the empirical run-time will be considerably better than the theoretical upper bound.

[4]In this tagger, the tag of a word is changed from X to Y only if the word was tagged with Y at least once in the training corpus.

where a particular operation can be triggered by a particular environment, then very little information can be gleaned about how likely it is that the transformation will improve performance on a new text. In general, the more instances we have to observe the effect of a transformation, the more information we have about the effect of the transformation on fresh text.[5] If the rank-frequency plot is relatively flat, meaning that there are few instances of all transformations being applied in the training corpus, then error-driven learning would probably not prove fruitful. However, we have found that for the error-driven learning systems we have examined, the rank-frequency plot is highly skewed. We will now turn to an examination of Zipf's law, an empirical observation that the rank-frequency plot of many different language-phenomena in many different languages, is highly skewed.

## 4.2 A Word on Zipf's Law

Zipf's law [116] is an empirical observation that in many different domains, the rank of an element divided by the frequency of occurrence of that element is constant. For instance, if city populations were to obey Zipf's law, that would mean that if the most populous city has population n, then the second largest city would have population n/2, the third largest n/3 and so on. Figure 4.4 (reproduced from [48]) demonstrates this phenomenon over actual city census data.

Zipf observed that this law seemed to hold for frequency data from a number of disparate areas, including city populations and word frequencies in texts written in various languages. He attributed this phenomenon to what he called the Principle of Least Effort [115, 116]. Subsequent to Zipf's claim of uncovering a universal property of human nature, a number of later publications demonstrated that Zipf's Law is a necessary consequent of assuming that the source of the language from which the frequency data is taken is a simple stochastic process [80, 105]. In the introduction to [115], George Miller elegantly puts it:

> Suppose that we acquired a dozen monkeys and chained them to typewriters until they had produced some very long and random sequence of characters. Suppose further that we defined a "word" in this monkey-text as any

[5] Assuming that the training text and the fresh text come from the same source.

| City | Rank | Population | (Rank) x (Population) x $10^{-6}$ |
|---|---|---|---|
| New York | 1 | 7,710,300 | 7.7 |
| Chicago | 2 | 3,511,000 | 7.0 |
| Los Angeles | 3 | 2,450,000 | 7.4 |
| Philadelphia | 4 | 1,971,200 | 7.8 |
| Detroit | 5 | 1,654,100 | 8.3 |
| Houston | 6 | 932,600 | 5.6 |
| Baltimore | 7 | 922,200 | 6.5 |
| Cleveland | 8 | 869,100 | 7.0 |
| St. Louis | 9 | 747,100 | 6.7 |
| Milwaukee | 10 | 732,600 | 7.3 |
| S. Francisco | 11 | 716,300 | 7.9 |
| Dallas | 12 | 672,400 | 8.1 |

Figure 4.4: Data from the 1960 Census

> sequence of letters occurring between successive spaces. And suppose finally that we counted the occurrences of these "words" in just the way Zipf and others counted the occurrences of real words in meaningful texts. When we plot our results in the same manner, we will find exactly the same "Zipf curves" for the monkeys as for the human authors.

If by "Zipf curve" we mean a highly skewed rank-frequency curve, then this statement is true. Assuming twenty-six characters plus space, then the probability of a particular *word* of length n is:

$$\frac{1}{27^{1+n}}$$

The monkeys will type 26 different words ("a", "b", ... "z") with probability $\frac{1}{27^2}$, $27^2$ different words ("aa", "ab", ... "zz") with probability $\frac{1}{27^3}$, and so on.

It is sufficient to note that empirically, Zipf's law seems to roughly hold for linguistic frequency data of many sorts across many different languages [114]. When plotting rank versus frequency in many different domains including words, word bigrams, part of speech bigrams and part of speech sequences of noun phrases, the resulting graph is highly skewed, with a few high frequency types accounting for a large percentage of total tokens, and a large number of types that occur very infrequently.

The distributional techniques to be explored in this dissertation all work by approximating the true distributional behavior of an element, or of the triggering environments for error-reducing transformations, from its observed behavior in a large corpus. The more instances we have of the element in the corpus, the more accurate will be our approximation. Because of this, Zipf's law tells us that we will have difficulty drawing any conclusions based upon distributional observation for most elements of the element type we are interested in (word, phrase, etc.). In addition to indicating that many elements will occur with very low frequency, we can deduce that there will be a great number of elements that are allowable, but do not occur in the corpus. This makes it difficult to know whether the nonoccurrence of an element in a corpus indicates that that element is not permitted in the language, or whether it is permitted but just does not occur in our sample corpus. Other than ignoring this problem, there are two approaches to dealing with it. The first is to use smoothing techniques to better approximate the probability of very low probability events. The second approach is to use distributional techniques that are less dependent on very low probability environments. The transformation-based learner takes the latter approach.

If we are concerned with accuracy as measured by tokens and not by types, then Zipf's law can work to our advantage.[6] Although only a small percentage of words that appear in a corpus appear with a high frequency, those high frequency words account for a large percentage of the total tokens in the corpus. Consider the following experiment. We take equal portions of French and English text, and then make a new text by repeatedly moving one word picked randomly from either text to the new text. Next we give the text to somebody who knows neither English nor French and ask them to take each word appearing in the mixed up text, and label the word as either being French or English. If the person picked randomly, they would be 50% correct. If we were to provide the person with a list of the 10 most probable words in both English and French, an accuracy of 63% would be obtained. If the word list was extended to 50 words, 71% accuracy would be possible. If instead the person was asked to build a dictionary listing which words appearing in the text are English and which are French, and accuracy was based upon the percentage of

[6]The sentence *the car ate the car* has 5 tokens and 3 types.

correct dictionary entries, then assuming a text size of one million words, giving the two lists of 50 words would give an accuracy of only 50%.[7]

To give a concrete example of Zipfian behavior in a natural language corpus, in the Brown Corpus[8][44], two percent of the word types account for sixty nine percent of the word tokens. About seventy five percent of the word types occur five or fewer times in the corpus. Fifty eight percent of word types occur two or fewer times, and forty four percent only occur once.

The rank-frequency plots of transformation number versus transformation score on the training set (and on the test set) are also highly skewed, which is one reason why the transformation-based learner is effective. In figure 4.5, the score received on the training corpus is plotted as a function of transformation number, showing a highly skewed Zipfian distribution. The transformations are from learning unknown word information on the Wall Street Journal. The experiment and the specifics of the score are described later.

The skewed rank-frequency curve for error-reducing transformations results in a number of properties of our learner. First, for a given training and test set there will be a long tail of very low frequency events in both corpora. Since these events are low frequency, it is likely that many will occur exclusively in only the training corpus or in only the test corpus. If an event (a triggering environment) occurs only in the training corpus, then harmless overtraining will result, since the system will learn transformations to remedy the error that is specific to that sample corpus. If it occurs in different form in one or the other, for example if a triggering environment leads to a beneficial transformation in one corpus and a detrimental one in another, then harmful overtraining will result. This problem could be resolved, or at least lessened, by using a second training corpus to prune transformations. However, because of the nature of the learner, overtraining is not as harmful in transformation-based learning as it is in other learning paradigms being applied to corpus-based learning (for instance, see [88, 7]). At every stage of learning, the transformation-based learner learns the transformation that results in the greatest error reduction. As can be seen in the graphs in later chapters, accuracy typically improves

[7]However, information at the character level – morphological information or probability information on character pair occurrences – could significantly increase the accuracy.

[8]A corpus of about 1 million words, containing samples from many different genres of written English.

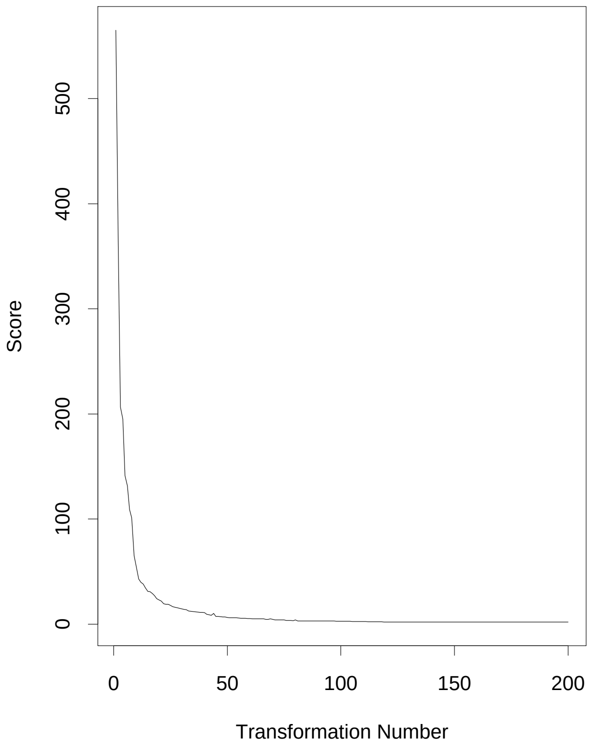



Figure 4.5: Zipfian Distribution Of Transformation Scores.

rapidly after applying the first few transformations, with the rate of improvement declining as later transformations are applied. Assuming that the training and test sets are generated by the same source, the probability of a transformation that results in high improvement on the training corpus being specific to a particular sample corpus is much smaller than the probability of this occurring for a low improvement transformation. Overtraining does occur, but it generally occurs in the low improvement transformations that do not contribute much to the final structure. The low improvement transformations that arise from overtraining do not necessarily result in performance degradation. Triggering environments for these transformations either do not occur at all in the test corpus, resulting in no change, or occur with very low frequency, typically resulting in random change if the transformation is due to overtraining and positive change if it is not.

If an event occurs only in the test corpus, then it is most likely a low-frequency event. Since it is low-frequency, incorrectly processing it will not result in a great performance degradation. On the negative side, if the events truly exhibit a Zipfian distribution, then doubling the size of the training corpus will roughly half the number of unseen events in the test corpus. This Zenoan behavior will result in a ceiling on achievable performance based on this method. With luck, this performance ceiling will be within the bounds of usable system performance. If not, then a more expressive set of transformations and more complicated search strategy can lift the performance ceiling. Note that the Zipf-like distribution is with respect to a certain descriptive language for describing events. It might be possible that once the learner is in the tail of the distribution, it can switch to a different descriptive language which would redistribute the residual errors in a way that makes them once again Zipf-like.

# Chapter 5

# An Overview of the Transformation-Based Learning System

The main goal of this dissertation is to propose a particular corpus-based learning algorithm and evaluate its effectiveness in learning structural information about natural language. When Harris and other structural linguists developed programs to aid the field linguist in uncovering structural information about language, they did not present the programs as language learning systems. With the advent of very fast computers and the availability of annotated on-line corpora, it is worthwhile reconsidering whether corpus-based learning algorithms can be made into real language learners. We have developed a learning algorithm which we believe to be quite successful at learning a considerable amount of structural information about language.

In building the programs that comprise the learning system, we follow Harris' layered approach [53], first addressing the learning of word classes and then learning phrase structure. This is done in part because mapping words into classes can help get around the sparse data problem in phrase structure learning.

We first describe a weakly supervised transformation-based error driven learning method for learning the necessary information to accurately tag words with an appropriate word class tag for a particular context. There are two steps in this process. First, lexical information is learned to guess the most likely tag for a word. A small corpus annotated with parts of speech and a larger unannotated corpus are used in training. For words seen in the annotated corpus, a lexicon is built indicating their most likely tag. The annotated and unannotated corpus are used to automatically learn a set of transformations that can be used for tagging words not covered by the lexicon. Second, contextual cues are learned for improving tagging accuracy. Next, a method is described for parsing text once word class information has been discovered. The parsing module is also based on error-driven learning. Parsing is also broken down into two steps: first, bracketing information is learned, and second, information is learned that can be used to label nonterminal nodes. Once structure is output by the parsing module, it can be fed into other modules to further decrease errors. One such module is a prepositional phrase attachment module which can be used to increase the accuracy of the learner on this task.

The final goal of this project is to train the different learning modules such that unannotated, free text can be assigned the proper structural annotation. Rather than aiming for structure that is *proper* in the platonic sense, we attempt to match the structure provided in a manually annotated corpus. This provides us with an objective way for evaluating the success of the different learning modules. Each module will be evaluated independently, but the effectiveness of the lexical modules can also be measured by the accuracy of the phrase structure modules that are trained on text annotated using transformations learned in the lexical modules.

The language learning modules that operate on various structural levels all share one thing in common. They all learn structure using the tool of transformation-based error-driven learning. In this learning paradigm, the system begins in a language-naive state.[1] The system then repeatedly compares its output to proper output and learns transformations to make its output better resemble correct output. In all cases, the set of allowable transformations is extremely simple. There are two parts to transformations: the transformation itself and the environment that triggers it. In all modules of this learner, the triggering environments are simple and astructural. For example, in the bracketing module,

[1]This is not a necessary property of the learning paradigm, but is true for all of the learning modules described in this dissertation.

a transformation is triggered by the part of speech of a single word or a pair of contiguous words. We have used simple transformations and small training corpora to try to increase the portability of the system. If a system can be built which makes use of no language-specific information and can be adequately trained on relatively small corpora, then the system can be easily used to annotate any corpus given only a little time for a person to annotate a small training corpus. It is the goal of this work to produce a system that can be readily adapted to a new task with minimal human supervision.

The field of corpus-based natural language processing is highly empirical. It is often difficult, if not impossible, to give formal explanations about the performance of a technique applied to a natural language corpus. This is in part because so little is known about the underlying structure of natural language. Therefore, the success of a method must be demonstrated empirically. One must be careful in how one attempts to empirically demonstrate the performance of a system.[2] If testing and training are only carried out on one corpus then we cannot know if we merely have a method that happens to be good on that corpus and only on that corpus. To partially address this concern, we have tested the learning procedure on a number of different corpora. We have tested across genre (Wall Street Journal, Brown Corpus and ATIS Corpus), across part of speech tags (Penn Treebank Tagging and Original Brown Corpus Tagging) and across languages (English and Old English)[83, 56, 44, 20].

We will now discuss the details of the different learning modules.

[2] We will not even touch upon certain obvious problems such as explicitly or implicitly training or developing on the test data, a methodology flaw which takes away the possibility of making any claims about a system capturing any generalizations.

# Chapter 6

# Lexical Information

In this chapter, we describe a method for tagging a large corpus or infinite stream of text, given only a small corpus ($< 45{,}000$ words) of tagged text and a large corpus of untagged text as training material. There are three steps in this process. First, a set of part of speech tags is found. This could probably be done manually fairly easily, but we provide a tool that can be used to aid a person in choosing a set of tags. This step is not a central part of the thesis, but it is interesting and useful nonetheless. The second step involves learning lexical information. In this step, a set of transformations (or rules) is discovered that can be applied to a word in order to find the word's most likely part of speech. It appears to be the case[1] that tagging every word in a corpus with its most likely tag will result in fairly high tagging accuracy. In this step we are learning information about word types. After trying to learn the most likely tag for words, we then learn a set of transformations that use contextual information to correct errors in tagging. This information is on the level of word tokens. For example, while we might have learned in the lexical phase that *can* is most likely a modal, in the contextual phase we might learn that a particular use of the word *can* appearing immediately to the right of the word *the* is most likely a noun.

Being able to reliably classify words is a necessary step toward automatically annotating a corpus with phrase structure. If phrase structure learning were done without using word classes, serious sparse data problems would arise from viewing a corpus merely as a string of words without ever abstracting away to more general classes. Mapping words into classes

[1] At least in the English and Old English corpora we have examined.

is a necessary generalization for overcoming sparse data. Likewise, a rule stating that a determiner and a noun combine to make a noun phrase would be much easier to learn than the large number of lexical-pair rules that would be required if word class information were not available. Providing words with preterminal syntactic labels can be useful information to have in a syntactic tree, and indeed is used by the transformation-based phrase structure learner which converts a syntactic tree with only preterminals labelled to a tree with all nonterminal nodes labelled.

A reliable part of speech tagger is also a useful tool in isolation. Part of speech tags can aid systems such as spelling correctors and speech recognition and generation systems. For instance, if a speech system is to properly pronounce the word *record*, it must know whether the word is being used as a noun or as a verb.

Since a number of fairly reliable part of speech taggers have been developed recently (e.g. [23, 36, 40, 45, 39]), one may ask why we bother exploring the possibility of creating a part of speech tagger with minimal human supervision. First, it has been shown that a tagger trained on one corpus will perform much worse on a different corpus. In [84], an experiment is run where a tagger is first trained on the Wall Street Journal and tested on the MUC corpus, a corpus of texts on terrorism in Latin America. Next, both training and testing were done using the MUC corpus. Training and testing on the same type of corpus resulted in a 34% reduction of error over training and testing on different types of corpora. If one wishes to use the precise tag set that an already-made tagger has been trained on and to apply the tagger to the exact same type of text as that used for training, then to use a tagger that is trained using minimal human supervision is unnecessary. However, if one wishes to use a different tag set, or apply the tagger to a different corpus, or even use the tagger for a different language, then the tagger will have to be retrained. Currently, training an accurate tagger requires a great deal of human labor. For example, in the tagger described in [36], the program includes:

- Statistics gathered from one million words of manually tagged text.

- Rules discovered via experimentation for dealing with hard tagging distinctions such as proper noun vs. common noun.

- A manually encoded list of dates, unlikely proper nouns, titles, states.
- A module for dealing with hyphenated words.
- Etc.

To retrain this tagger for use on a significantly different corpus would be extremely tedious. In addition to requiring a large amount of manually tagged text, any of the additional rules that turn out to be corpus-specific would have to be rewritten. Instead, we propose a tagger which is very easy to train: a much smaller annotated corpus is needed for training and a procedure automatically learns appropriate transformations for the corpus being tagged. No corpus-specific or language-specific information need be specified. This means that the cost in terms of human effort needed to retrain the tagger to be used with a different tag set, corpus, or language is minimal.

## 6.1 Word Classes

Before a tagger can be built, a tag set must be specified. There is strong evidence that the set of possible classes which can be distinguished in a language is unbounded. Sapir thought that only the classes of noun and verb were fundamental to language. He wrote ([101], quoted in [96]): "No language wholly fails to distinguish noun and verb, though in particular cases the nature of the distinction may be an elusive one. It is different with the other parts of speech. No one of these is imperatively required for the life of language." Lakoff [74] describes a language in which the class $\pm$*woman-or-fire-or-dangerous-thing* exists. This class is based upon ancient folklore of the society in which it is used. In processing a corpus in an automobile subdomain, it might make sense to specify the class of automobile names, whereas in a general text such a specific class may be inappropriate. If the set of possible word classes is indeed unbounded, then it cannot be prespecified in a truly portable natural language processing system. The classes of a language, and particularly of a sublanguage, must be learned.

In this section, we will demonstrate a semi-automatic method for determining a set of appropriate word classes for a particular corpus.[2] Note that we are not classifying

[2] It is not certain that the method described in the following subsection for determining the set of classes

words at this stage, but are merely finding a set of classes into which words will later be assigned. There are two other paths that could be pursued, but which we have chosen not to pursue: (a) fully automatic word class discovery and (b) not decoupling word class discovery from word classification. We have chosen against pursuing (a) because the amount of manual labor necessary in the semi-automatic method is so small, we do not see a need for a fully automatic system. Approach (b) has been attempted elsewhere [29]; we believe that decoupling has the advantage of intelligently using a small amount of human supervision to guide the learning process in a way that should lead to more intuitive classes. One possible disadvantage to our approach is that if the evaluation measure is something possibly counterintuitive to humans, such as the set of classes that results in the greatest reduction of entropy, adding human intuition may mislead the learner.

The goal of this learning module is to aid the human in choosing a set of part of speech tags. A word similarity tree is built for the most frequently occurring words in the corpus. The tree is built by initially making every word a node and then repeatedly combining the most similar pair of nodes into a single node until all words are reduced to just one node. Regions in the tree will tend to correspond to word classes. Therefore, looking at such a similarity tree can help a person decide upon a set of appropriate part of speech tags.

The work is based upon the hypothesis that whenever two words are syntactically or semantically dissimilar, this difference will manifest itself in the syntax via lexical distribution, an idea suggested in [53]. We have made this idea amenable to automation by assuming that there is enough distributional information in local and astructural environments to accurately characterize the distributional behavior of a word.[3] In particular, information about the probabilities of words occurring immediately before and after a particular word is all that is used for distributional characterization. A number of different similarity measures could be used. We chose to use relative entropy, also known as the Kullback-Leibler distance [72, 38]. The Kullback Leibler distance from probability distribution P to probability distribution Q is defined as:

is necessary, as it might not accomplish anything that could not be done rapidly by human introspection. However, for a human to do this from scratch would probably require some linguistic knowledge as well as some familiarity with the corpus being processed.

[3] Earlier versions of this work appear in [22, 15].

$$D(P||Q) = \sum_x P(x) log \frac{P(x)}{Q(x)}$$

The divergence of P and Q is then defined as:

$$Div(P,Q) = Div(Q,P) = D(P||Q) + D(Q||P)$$

For two words x and y, let $P_{left,x}(z)$ be the probability of word z occurring immediately to the left of x. $P_{right,x}(z)$, $P_{left,y}(z)$ and $P_{right,y}(z)$ are defined likewise. We can then define the similarity of x and y as:

$$Sim(x,y) = 1 - \frac{Div(P_{right,x}, P_{right,y}) + Div(P_{left,x}, P_{left,y})}{2}$$

$Sim(x,y)$ ranges from 0 to 1, with $Sim(x,x) = 1$. A problem is encountered if any probability estimates are zero. To get around this, a small percentage of the probability mass is redistributed to ensure that all probability estimates are greater than zero.

To build a similarity tree, we do the following. Initially, every word is its own node. Then we repeatedly combine the two most similar nodes into one node, until only one node remains. Node similarity is defined as the average similarity between words in the two nodes.

Of the 300 most frequently occurring words in the Brown Corpus, figure 6.1 shows the thirty pairs deemed closest distributionally, using the *Sim* measure. When we found classes of words in the Brown Corpus, word pairs seen fewer than three times are considered to have probability zero. Of all word bigrams in the Brown Corpus, only 14.5% occur with frequency greater than two. We have found that ignoring low frequency bigrams, while greatly reducing the computation time, does not seem to affect the word pair similarity results. This means that issues such as providing good probability estimates for observed frequencies of zero need not be addressed.

The same experiment was run on a corpus of roughly 1.7 million words of transcribed utterances addressed by parents to their young children [78]. The thirty most similar pairs in that corpus are listed in figure 6.2. From this list we can see that although the word lists are different, the method was effective in both cases at grouping word pairs together

| | |
|---|---|
| HE SHE | COULD CAN |
| WE THEY | BUT ALTHOUGH |
| GIVE MAKE | WHILE ALTHOUGH |
| ME HIM | KIND NUMBER |
| IF WHEN | FIND TAKE |
| GET TAKE | ALTHOUGH SINCE |
| FIND MAKE | GET MAKE |
| THEM HIM | WHEN ALTHOUGH |
| IF THOUGH | MADE FOUND |
| MAKE TAKE | MEN CHILDREN |
| GIVE TAKE | MUST SHOULD |
| MEN PEOPLE | US THEM |
| FACE HEAD | CAME WENT |
| GET FIND | GIVE GET |
| SENSE KIND | TIME DAY |
| COULD WOULD | MIGHT MAY |

Figure 6.1: Word Similarities from the Brown Corpus

that share features. It is significant that these results are obtained on the parental speech, as this corpus is much noisier than the Brown Corpus, containing many false starts, typos, fragments and run-ons.

The experiment was also run on the Voyager Corpus, a corpus consisting mainly of short questions about Cambridge and Boston. The version of the corpus we used had fewer than 40,000 words total. The 125 most frequently occurring words were chosen and a similarity tree was built for these words. Figure 6.3 shows the thirty word pairs which are most similar distributionally in this corpus.[4] As an example of sublanguage classes, note that *walk* and *get* are considered very similar in the Voyager Corpus, whereas we would not consider these two words to be similar in normal unconstrained language. In the Voyager Corpus, *get* is not used to mean procure, but rather to mean *get from point A to point B.* This sense of *get* is synonymous with *walk*.

Although this method looks somewhat promising, it has not been shown how to extract a useful set of classes automatically from the resulting similarity trees. There have been

[4] Kendall, Central, Harvard and Inman are all squares. Dolphin and Legal are restaurants. Marriott and Charles are hotels. Pearl and Magazine are streets. Massachusetts and Western are avenues.

| | |
|---|---|
| YES YEAH | MARK ROSS |
| UHHUH MMHM | SHOW TELL |
| HE SHE | DOOR TABLE |
| OKAY ALRIGHT | THOSE THESE |
| YEAH YEAH/YES@Q | OH WELL |
| OK OKAY | HEAD MOUTH |
| UHHUN YEAH/YES@Q | OH YEAH |
| OK ALRIGHT | UHHUH YEAH |
| YEAH OKAY | OK YEAH/YES@Q |
| YES YEAH/YES@Q | OH YES |
| OK YEAH | YEAH/YES@Q MMHM |
| BECAUSE CAUSE | YES NO |
| PUT TAKE | BRING GIVE |
| COULD CAN | THERE HERE |
| YEAH ALRIGHT | MOUTH NOSE |

Figure 6.2: Word Similarities on Parental Speech

| | |
|---|---|
| KENDALL CENTRAL | WHAT WHERE |
| SHOW TELL | INMAN CENTRAL |
| DOLPHIN LEGAL | INMAN HARVARD |
| MARRIOTT CHARLES | PEARL MAGAZINE |
| INTERSECTION ADDRESS | FROM TO |
| WALK GO | WHAT WHICH |
| CLOSEST NEAREST | LIBRARY BAYBANK |
| WALK GET | MASSACHUSETTS WESTERN |
| GET GO | MAGAZINE BROADWAY |
| KENDALL HARVARD | DO COULD |
| CENTRAL HARVARD | WHICH WHERE |
| COULD CAN | MIT LAGROCERIA |
| DO CAN | IN NEAR |
| INMAN KENDALL | IS ARE |
| STATION SQUARE | FAR LONG |

Figure 6.3: Word Similarities on Voyager Corpus

a number of other attempts at automatic word classification using an approach similar to that described in this section. In [60], words are classed according to their distribution in subject-verb-object relationships. For such a method to succeed, one would need to be able to accurately parse the text being analyzed prior to word classification. The classification method we described above requires no structural information. [97] and [70] both attempt to classify words based upon their immediate neighbors. They use a similar definition of environment as used in our system, but use different measures of similarity. [97] ran a small-scale experiment, running the learning procedure over 105 basic English sentences from a simple introductory English language text containing a total of 131 different words. The method proposed by Kiss is not fully automatic. He manually chooses the set of words that will be clustered. The experiments of [97] and [70] left open the question of whether these techniques could succeed on free text.

[29] describes another method of classifying words based upon distributional similarity of words in adjacent-word environments. They attempt to find the assignment of words to classes which results in the smallest loss of average mutual information between immediately adjacent word classes in the corpus. There are a number of important differences between that algorithm and the algorithm we have presented. For one thing, in our algorithm words are always compared based upon their distributional similarity with respect to adjacent words. In their algorithm, only the first two words grouped together are compared in this way. All other words are compared over a corpus where some words have been reduced into word classes. Mapping words into classes has the benefit of making sparse data less of a problem, but it also makes the distributional comparisons less precise. Since our method only uses high frequency observations, sparse data is not a problem. In their algorithm, classification is sensitive to word frequency. They are calculating the reduction in average mutual information. Therefore, two high frequency words may be grouped together before two lower frequency words that are more similar, if so doing results in a greater average mutual information. In our system, all word pairs in the list of $n$ most frequently occurring words are weighed equally. In addition, our method is computationally less expensive. We only compute once the divergence of words in high-frequency environments. They calculate the mutual information of the entire corpus, and must recalculate

this every time a pair of words is mapped into one class.

While all of these approaches to clustering seem to indicate that there is a great deal of information to be gleaned from local, astructural environments, it is not clear whether this approach can outperform an approach guided by a small amount of human supervision. It is important to emphasize that we do not believe that the clustering method we have described above will succeed by itself in finding a useful set of word classes, nor in correctly assigning all of the words in a corpus into classes. Rather, we view the clustering procedure as a way of eliciting the classes that are salient in a corpus. Once these classes are established, another procedure (such as that outlined in the following chapter) can be used to assign words to classes.

After the similarity tree has been automatically created, the word classes relevant to the particular corpus will correspond to particular regions in the similarity tree. We do not expect this procedure to result in a tree with only meaningful areas, but rather a tree which can be used as an aid to a human to glean a useful set of word classes for a corpus. Once a set of classes has been semi-automatically derived, the next step is to learn the classes each word can belong to, along with the rules governing word class disambiguation. A method for accomplishing this will be described in the following section.

In addition to aiding in the creation of a set of part of speech tags, the similarity trees can serve another function. In the part of speech and phrase structure learning modules, sparse data is not a problem. However, sparse data can be a problem in certain modules that employ transformation-based learning. For instance, in the prepositional-phrase-attachment module described later, transformations make reference to the head of a noun phrase or verb phrase. Since we can assume that the head of a noun phrase will be a noun, part of speech tags provide no additional information here. If transformations only make reference to particular words, sparse data problems may be encountered. One solution to this problem would be to use a manually created lexical hierarchy such as Wordnet[85], and allow transformations to make reference to words and to the word classes each word belongs to. A drawback of this approach is that manually created hierarchies are expensive and time-consuming to create. An alternative method of avoiding sparse data involves first creating a distributional similarity tree for all nouns in a corpus. Then a unique feature name could be associated with all nodes in the tree. For each word and feature x, the word will be $+x$ if it is a descendent of the node labelled with feature name x, and will be $-x$ otherwise. Transformations are then allowed to make reference to these class names as well as to particular words. If this results in too many classes, we can instead ignore all nodes except those of distance d from the root for which $d \bmod n = 0$ for some appropriate n.

## 6.2 Finding the Most Likely Tag for Each Word

The next step in building the tagger is to try to find the most likely tag for each word. There are a number of ways to view this problem. To be concrete, we will address the following specific problem: given a small tagged corpus and a much larger untagged corpus, try to accurately tag the large untagged corpus. For example, if one had a corpus of ten million words to tag, an informant could first tag a small subset of that corpus, and then the learning procedure could be used to tag the rest. For words in the large untagged corpus that also occur in the tagged corpus, we can initially tag them with their most likely tag as indicated in the tagged corpus. For the other words, we can learn cues to help us automatically guess their most likely part of speech. To do this, we will use transformation-based error-driven learning.

Unknown words are a big problem in part of speech tagging, especially when building a lexicon from a very small corpus. In figure 6.4, we show a graph of the percentage of tokens in a test set not in the lexicon built from the training set for various sized training corpora of the Wall Street Journal. When a lexicon is built from more than three million words of text, then fewer than 3% of word tokens in new text were not in the training text. However, when a lexicon is built from a corpus of 7,280 words, 36% of the word tokens in new text were not included in the training text.

A number of different part of speech tagging systems have addressed the problem of unknown words. Since we are interested in a system that can be easily trained and retrained for new domains or languages, we will not discuss methods that have a great deal of domain-dependent knowledge built in (such as [36, 45]). In [73, 84], a probabilistic method of tagging unknown words is discussed. Since the two methods are fairly similar,

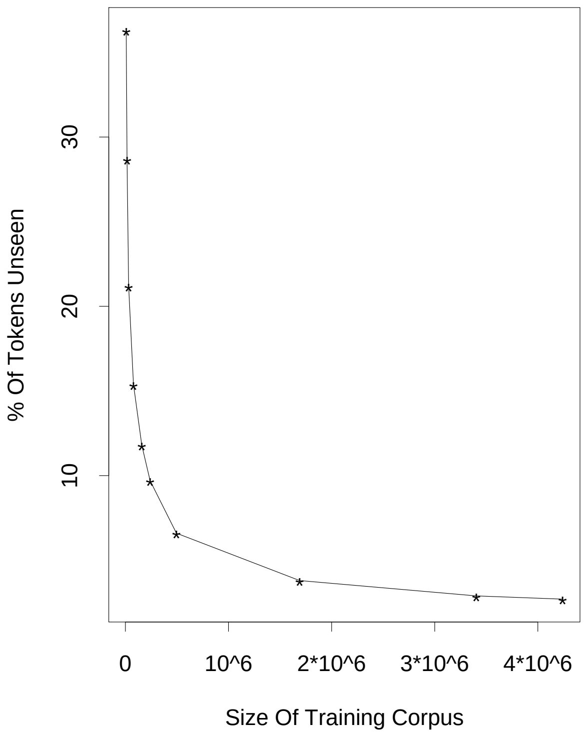



Figure 6.4: Unknown Words vs Training Corpus Size

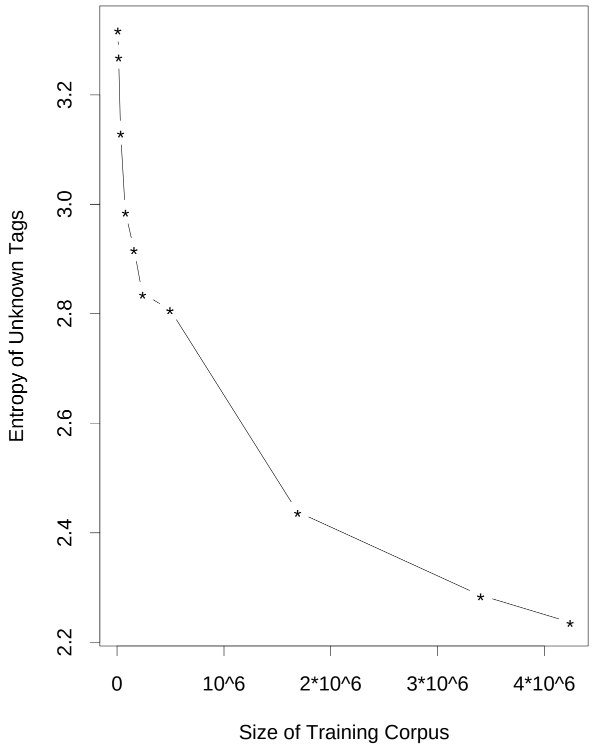



Figure 6.5: Entropy of vs Training Corpus Size

we will only describe the algorithm presented in [84]. The tagger used is a Markov-model based tagger trained on tagged text. For known words, $P(W|T)$ is estimated from the corpus. Since they assume there will be a large training corpus, they make the assumption that unknown words can only be tagged with one of the 22 open-class Penn Treebank part of speech tags. Note that we cannot make that assumption in our system, since our training set is much smaller and may not cover all closed class words. The entropy of the tag distribution for unknown words will be much greater if a very small corpus is used to build a lexicon than if a large corpus is used. In figure 6.5, the entropy of the tag distribution for unknown words $(-\sum_{X \in Tags} P(X|Unknown) * log(P(X|Unknown)))$ is graphed as a function of training corpus size for the same Wall Street Journal samples used in the previous figure.

In [84], unknown words are first addressed by attempting to tag using a trigram model by adding $P(\text{Unknown Word}|T)$ for all open class tags T, with training carried out on about one million words of tagged text from the Wall Street Journal.[5] In other words, only general lexical information is used regarding the class of unknown words as a whole, and contextual probabilities combined with this general lexical information is then used to disambiguate. This results in an accuracy of 48.4% on tagging unknown words. They then try using the following lexical probability for unknown words:

$$P(w_i|t_j) = P(\text{Unknown Word}|t_i) * P(\text{Capital Feature}|t_i) * P(\text{Endings/Hyphenation}|t_i)$$

where *Capital Feature* is one of the four possible settings of $(\pm initial, \pm capitalized)$, and the endings are from a set of 35 manually chosen suffixes. This results in an accuracy of 85% using the large training corpus. We will demonstrate below that the transformation-based approach significantly outperforms this statistical method both on tagging unknown words and overall when both are trained on a much smaller training corpus and obtains comparable performance overall on large corpora.

In [24], a different statistical approach was taken to determining the class of unknown words. An informant first listed the open class tags for the corpus. Next, for each open class tag a small list of exemplar words (5-8) was given. These exemplar words were used to build a *distributional fingerprint* for each open class: a vector of the probability of words appearing immediately before (and after) any of the class exemplar words in the corpus. Unknown words are then classified by comparing their distributional fingerprint to that of each open class and assigning it to the class which is most similar. The similarity measure used was relative entropy [72]. We have found that the transformation-based approach significantly outperforms this distributional approach for classifying unknown words.

In the transformation-based system, the lexicon will initially be built from the small manually annotated corpus. Since we want to keep this corpus as small as possible to minimize the amount of work a person needs to put in, we will not be able to ignore the issue of encountering words that do not appear in our lexicon.

In its initial state, the transformation learner assumes all never before seen words have the same default label as their most likely part of speech tag. The default label is set to

[5] While they do not explicitly state the training corpus size for these experiments, one can figure this out from their table of error rates (overall, for known words and for unknown words), along with knowledge of the percentage of unknown words as a function of corpus size.

the most frequently occurring tag, measured by word types, in the training corpus. A set of transformation templates is prespecified, defining the types of cues that can be used to indicate that the assumed most likely tag for a word should be altered. Currently, the templates are:

- Change the most likely tag to X if deleting the prefix x, $|x| \leq 4$, results in a word.
- Change the most likely tag to X if the first (1,2,3,4) characters of the word are x.
- Change the most likely tag to X if deleting the suffix x, $|x| \leq 4$, results in a word.
- Change the most likely tag to X if the last (1,2,3,4) characters of the word are x.
- Change the most likely tag to X if adding the character string x as a suffix results in a word ($|x| \leq 4$).
- Change the most likely tag to X if adding the character string x as a prefix results in a word ($|x| \leq 4$).
- Change the most likely tag to X if word Y ever appears immediately to the left/right of the word.[6]
- Change the most likely tag to X if character Z appears in the word.
- All of the above transformations, modified to say Change the most likely tag from Y to X if ...

The templates could be extended to handle languages with infixes by allowing a transformation such as: *Change a tag if the character string X appears internal to a word.*

Note that in all of these transformation templates, the trigger can be determined from an unannotated text. Therefore, whether the trigger applies for a particular word type is computed based on the words and word pairs occurring in the large unannotated training corpus. Note also that while bigram statistics have been used often to characterize words (eg. [22, 15, 24, 29, 37]), the approach taken here is different in that unlike all of these

[6] For reasons of processing efficiency, Y is constrained to be one of the n most frequently occurring words, where n was arbitrarily set to 200 in all experiments described here.

systems we are not using any statistical information about word pair cooccurrence. The only information used is the boolean value of whether a particular bigram was seen *at all* in the training corpus. This is similar to the nonstatistical use of distributional environments in the theory of Harris [53]. Harris states that since using the sum total of all allowable short environments an entity is licensed to appear in may not be a good way to classify words, a linguist could find diagnostic environments that can be used to test if a word belongs to a particular class. We are in essence providing an automatic procedure for discovering diagnostic environments.

In order for the transformation learner to be fully specified, we must now define the evaluation measure used in the search for an ordered list of transformations. We have a small annotated corpus, and we can use this corpus to measure the effect of carrying out a particular transformation. We have to be somewhat careful in how the results of applying a transformation are evaluated. If we were to evaluate results on a per token basis, results would be skewed in favor of the higher frequency words. However, it is unlikely that many high frequency words will have to be treated as unknowns, since the probability of these words occurring in any training corpus is relatively high. To avoid this problem, success is measured on a per-type basis instead of a per-token basis. This means that the effect of a transformation on the tagging of the word *the* will be given equal consideration as the effect on the word *upside-down*.

A transformation that says to change the most common tag for a word from X to Y if trigger Z holds for that word is given the score:

$$\sum_{\text{W = Word in annotated corpus}} \frac{Freq(W,Y) - Freq(W,X)}{Freq(W)} * Z(W) * Current(W,X)$$

where

$$Z(W) = \begin{cases} 1 & \text{if trigger Z holds for word W} \\ 0 & \text{otherwise} \end{cases}$$

and

$$Current(W,X) = \begin{cases} 1 & \text{if W is currently tagged with X} \\ 0 & \text{otherwise} \end{cases}$$

and the frequencies are calculated from the small manually tagged corpus.[7] This function measures the per type improvement that results from carrying out the given transformation. To learn an ordered set of transformations, we find the transformation with the best score, add that transformation to the list, apply the transformation to the corpus, find the best transformation for the transformed corpus, and so on until no more transformations can be found or the score of the best transformation drops below some threshold.

The run time of the learning algorithm is $O(|op| * |env| * |n|)$, where $|op|$ is the number of allowable transformation operations, $|env|$ is the number of possible triggering environments, and $|n|$ is the training corpus size (the number of word types in the annotated lexical training corpus). Fortunately, we do not actually have to apply every possible transformation at each learning iteration. Learning is data driven. The theoretical upper bound on the number of transformations that have to be tested is significantly greater than the number of transformations whose examination will be triggered by their occurrence in the corpus. As an example, in one Wall Street Journal sample of 1,000 sentences there are 44 part of speech tags. If tag pairs trigger transformations, then $44^2 = 1936$ transformations would have to be examined if the search were not data driven. In reality, only 712 part of speech tag bigrams occur in this sample corpus. In the same sample of text, there are 74 unique characters, and therefore $74^4 = 3\text{x}10^7$ possible suffixes of length 4. Only 1781 suffixes of length 4 actually occur in this sample corpus.

Figure 6.6 shows a short Perl [110] pseudocode program that iteratively finds the best transformation, assuming the only allowable transformation template is:

Tag a word as X if the last letter is __

Because the method of learning is essentially the same for all modules, we will describe this pseudocode in some detail. While transformations can be found with a positive score, we search for the best transformation given the current state of the corpus. For each possible tag and each possible word, we update the score for changing to that tag given the last letter of the word being examined. After this scoring is completed, the scores for all (tag, last letter) pairs are examined, and the pair with the best score is recorded as the best transformation. This transformation is then applied to the corpus, and the process

[7] $Freq(W, X)$ is the number of times W is tagged with X in the training corpus.

continues. The procedure is data-driven in the sense that while we theoretically have to check all (tag, last letter) pairs, we really only do this for pairs containing a last letter actually occurring in the corpus. The difference between theoretical and actual run time will be more significant when dealing with more sparsely distributed phenomena, such as the last three or four letters of a word.

After an ordered list of transformations has been learned, the transformations are applied in order, to words in a fresh text (test corpus) that do not occur in the training corpus, using a large unannotated corpus to determine whether a transformation trigger applies to a particular word. Since an upper bound can be placed on the run-time of carrying out a single transformation, run time is linear with respect to the size of the unknown word list for the corpus being annotated for a given transformation list. As a function of transformation list size $|T|$ and unknown word list size $|n|$, run time is $O(|T| * |n|)$.

### 6.2.1 Results

To assess the accuracy of this learning module, we tested it on two different English corpora and one corpus of Old English.[8] Three different part of speech sets were used: Penn Treebank tags for the Brown Corpus and Wall Street Journal[83], the original Brown tag set for the Brown Corpus[44], and a tag set which is a derivative of the Penn Treebank tags derived by Eric Haeberli and Tony Kroch for the Old English corpus.

#### Wall Street Journal

The Wall Street Journal corpus is a set of stories from the Wall Street Journal sorted in chronological order. The corpus was used as follows: there are a total of 55,787 sentences and 1,340,777 words.[9] Training was done on the first 50,000 sentences and testing was done on the last 2787 sentences. This provided a buffer of 3,000 sentences between the training and testing set, which should be enough to minimize proximity effects. Lexical part of speech information was learned from an annotated subset of the corpus, the lexical training corpus, which consisted of the first 1,000 sentences (23,413 words) of the corpus. Contextual

[8]Thanks to Eric Haeberli and Tony Kroch for annotating the Old English corpus.
[9]The *combined* files, created by Rich Pito, were used.

```
$bestguy="";
# bestguy stores the best transformation.
$bestscore = 0.1;
# bestscore stores the score for the best transformation.
# Start with an arbitrary nonzero setting to get the program going.
@taglist = (NN,NP,...)
# a list of all part of speech tags
@wordlist = <STDIN>;
# the annotated training corpus word list.
while($bestscore) {
    $bestrule="";
    $bestscore = 0;
    foreach $tag (@taglist) {
        undef %transformation;
        foreach $word (@wordlist) {
            @characters = split(//,$word);
            $lastletter = $characters[$#characters];
            # Get the last letter of the word.
            $lastlettertransformation{join(" ",$tag,$lastletter)} +=
                &score($word,$tag); }
        # Alter the score for the transformation involving the currently
        # processed tag and last letter.
        while(($key,$val) = each %transformation) {
        # Go through all recorded transformations and find
        # the one with the best score.
        @temp = split(/\s+/,$key);
        $tag = $temp[0];
        $letter = $temp[1];
        if ($val > $bestscore) {
          $bestscore = $val;
          $bestrule = "Change to $tag if last letter is $letter\n";}}}
print "++$RULENUMBER $bestrule \n";
%updatecorpus; }
```

Figure 6.6: Perl Pseudocode For Learning Simple Transformations.

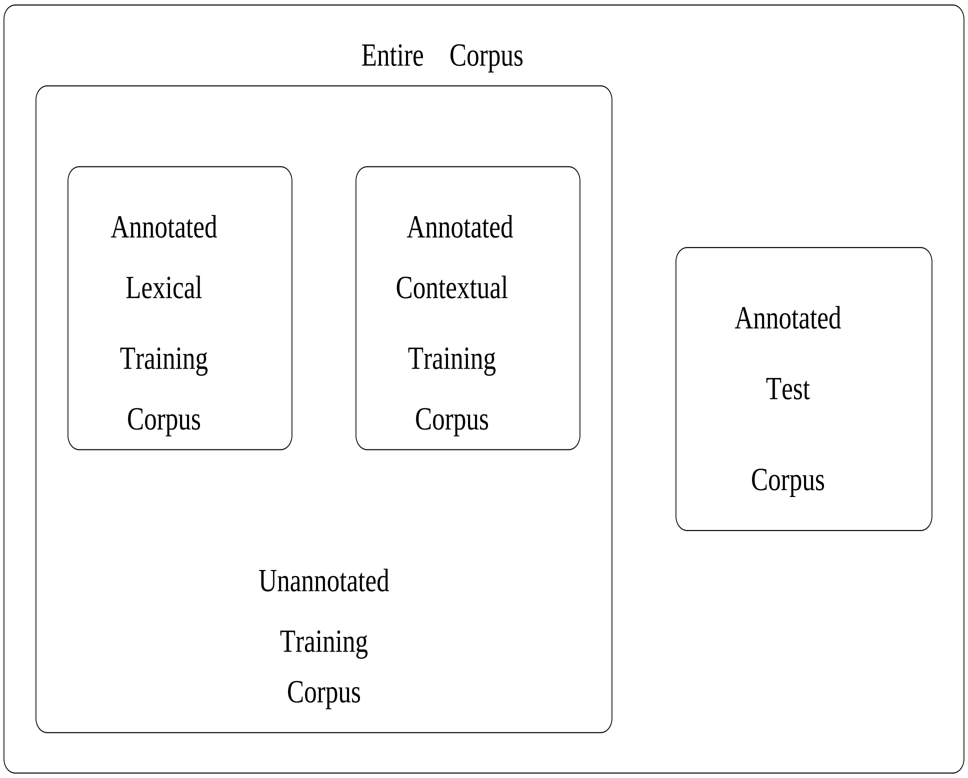



Figure 6.7: Dividing Up The Corpus For Running Experiments.

information was later learned using the contextual training corpus, which consisted of the second 1,000 sentences of the corpus (see figure 6.7).

First, a lexicon was built indicating the most likely tag for words in the annotated lexical training set (the first 1,000 sentences). 81.3% of the tokens in the test set appear in the training set, and tagging these words with their learned most likely tag results in an accuracy of 93.6%. Unknown words are initially assumed to be singular nouns. This gives a tagging accuracy for unknown words (measured on tokens, not types) of 19.2%. Transformation scores are computed using the annotated lexical training corpus, while trigger information such as whether a particular string is a word is based on the entire unannotated training corpus. In total, 200 transformations were learned. In figure 6.8 we list the first thirty learned transformations.[10] The first transformation states that the most likely tag for any word ending in *s* should be changed to plural noun. The second transformation states that the most likely tag for a word should be changed from a common

[10] See appendix 1 for a listing and description of the Penn part of speech tags.

noun to a proper noun if it is ever seen in the large unannotated corpus at the beginning of a sentence. When the second transformation applies, all words are tagged as either common nouns (the default) or plural nouns (the result of the first transformation). The second transformation only applies to singular common nouns, since plural common nouns can appear at the beginning of a sentence. The interaction of transformations number 1 and 27 is interesting. These transformations combined state that a word whose last letter is *s* and whose second to last letter is not *s* should be tagged as a plural noun. There are two obvious transformations that are approximated in the learned transformations:

- If a word begins with a capital letter, then it is a proper noun.
- If a word contains any of the characters [0-9], then the word is a number.

However, the transformation templates currently used are not sufficiently expressive to capture this information concisely.

Each of the 200 transformations was applied to the list of unknown words in the test corpus, with accuracy improving from an initial tagging accuracy of 19.2% when all unknown words are tagged as singular common nouns to a final accuracy of 77.5%. A graph of accuracy as a function of transformation application number can be seen in figure 6.9.

We next investigated the possibility of pruning the transformation list. The transformations were pruned by applying them to a second training corpus, and deleting all transformations whose application resulted in a lower accuracy. Doing so resulted in 163 transformations, whose application to the test set resulted in an accuracy of 77.3%, slightly lower than the unpruned transformation list. The results from applying the pruned transformations to the test corpus can be seen in figure 6.10. When applying the unpruned list, 97 transformations result in performance improvement, 50 result in performance degradation and 53 result in no change. When applying the pruned list, 91 transformations result in performance improvement, 21 result in performance degradation, and 51 result in no change. Although the pruned list results in marginally worse performance, fewer bad transformations are found. Unfortunately, a number of effective transformations are also deleted when pruning.

In all of the experiments done on training a word classifier, a threshold was chosen such

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | ?? | NNS | Suffix is *s* |
| 2 | NN | NP | Can appear at the start of a sent. |
| 3 | ?? | VBN | Suffix is *ed* |
| 4 | ?? | CD | Can appear to the right of *$* |
| 5 | ?? | VBG | Suffix is *ing* |
| 6 | ?? | JJ | Character - appears in the word |
| 7 | ?? | JJ | Adding the suffix *ly* results in a word |
| 8 | ?? | RB | Suffix is *ly* |
| 9 | ?? | NP | Can appear to the right of *Mr.* |
| 10 | NN | VB | Can appear to the right of *will* |
| 11 | NN | CD | Character *1* appears in the word |
| 12 | NN | JJ | Can appear to the right of *be* |
| 13 | NN | NP | Character *S* appears in the word |
| 14 | ?? | NP | Character *M* appears in the word |
| 15 | VB | NN | Can appear to the right of *the* |
| 16 | ?? | NP | Character *C* appears in the word |
| 17 | NNS | VBZ | Can appear to the right of *it* |
| 18 | ?? | NP | Character *B* appears in the word |
| 19 | NN | NP | Character *A* appears in the word |
| 20 | ?? | NP | Prefix is *D* |
| 21 | NN | NP | Character *H* appears in the word |
| 22 | NN | NP | Character *P* appears in the word |
| 23 | NN | JJ | Suffix is *c* |
| 24 | NN | CD | Character . appears in the word |
| 25 | ?? | NP | Prefix is *G* |
| 26 | NN | NP | Character *W* appears in the word |
| 27 | NNS | NN | Suffix is *ss* |
| 28 | NP | IN | Can appear to the left of *his* |
| 29 | JJ | NN | Can appear to the left of *has* |
| 30 | ?? | NP | Can appear to the left of *Corp.* |

Figure 6.8: The first 30 transformations from the WSJ Corpus.

that transformation learning continued only if the score of the last learned transformation met the threshold. This threshold was set to 2 for all experiments in this chapter. The threshold was chosen prior to any testing, and was picked somewhat arbitrarily. A higher threshold has the advantage of only learning transformations with high probability of being useful, thereby lessening the amount of overtraining. A higher threshold also has the advantage of speeding up run time. The rank-frequency distribution of errors is highly skewed. For instance, of the 200 transformations learned for the Wall Street Journal, there are 103 transformations with scores in the range $[2, 3)$, 22 in the range $[3, 4)$, and 9 in the range $[4, 5)$. However, a higher threshold could hurt performance by throwing away effective low frequency transformations. If instead of a threshold of 2, the threshold is set to 3, then an accuracy of 78.0% from only 97 transformations is obtained using an unpruned transformation list, a slight improvement in performance, but a significant reduction in the number of transformations. The threshold could be automatically determined by running the trained annotator on a held out training set with the threshold set to zero, and then determining the threshold value that results in the highest accuracy.

Next, we used both annotated training corpora to train the lexical tagger to explore the effect of training corpus size on accuracy. This in effect doubled the size of the lexical training set (from 1,000 sentences to 2,000 sentences). Using the larger training corpus increased the percentage of known words in the test set from 81.3% to 86.5%. Known word accuracy remained at 93.6%. 228 transformations were learned for tagging unknown words, resulting in an unknown word tagging accuracy of 79.9% (compared to 77.5% when trained on 1,000 sentences). Total accuracy using only type-based lexical information rose from 90.5% to 91.7%, in part due to the more accurate tagging of unknown words and in part due to the lower percentage of unknown words in the test set. Once again doubling the size of the lexical training corpus to 4,000 sentences resulted in an unknown word tagging accuracy of 81.3%, and a total accuracy of 92.2%. See table 6.1. Keep in mind that these results are obtained prior to using any token-based contextual information.

We then compared our results to the results cited in [84] for tagging unknown words, using the lexical probabilities:

$$P(w_i|t_j) = P(\text{Unknown Word}|t_i) * P(\text{Capital Feature}|t_i) * P(\text{Endings/Hyphenation}|t_i)$$

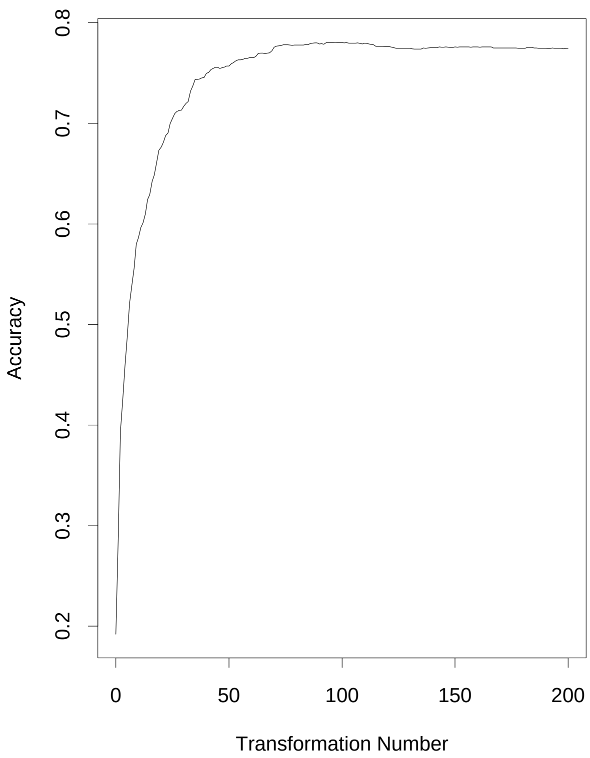



Figure 6.9: Unknown Word Tagging Accuracy After Applying Transformations.

| Training Corpus Size (Sents.) | Unknown Wd. Accuracy | Total Accuracy |
|---|---|---|
| 1,000 | 77.5 | 90.5 |
| 2,000 | 79.9 | 91.7 |
| 4,000 | 81.3 | 92.2 |

Table 6.1: Initial Tagging Accuracy as a Function of Training Corpus Size.

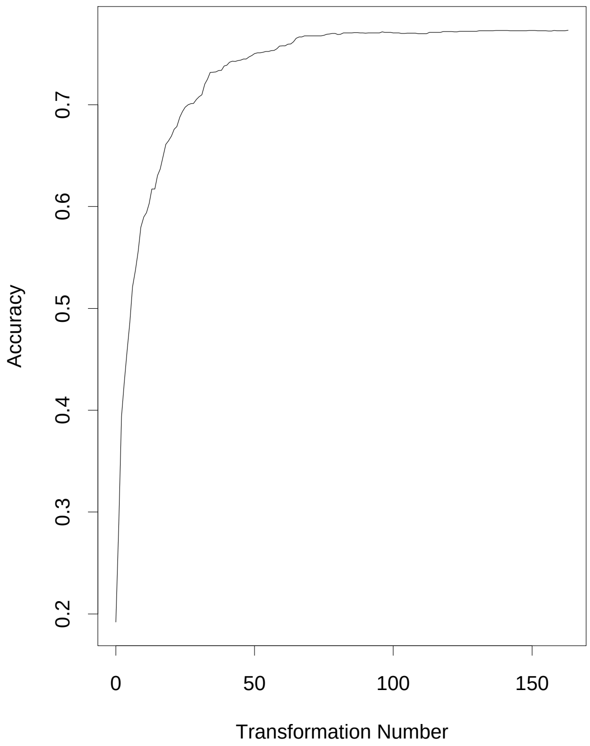



Figure 6.10: Unknown Word Tagging Accuracy After Applying Pruned Transformations.

for unknown words, as described above. They state that 35 suffixes were used in their system, of which seven are listed in the paper. We first implemented their algorithm using the suffixes they list in their paper. Testing and training were carried out on the same corpora as were used for the above experiments. First, using only $P(unknown|T)$ for lexical probabilities, an accuracy on unknown words of 51.8% is obtained. Using the more sophisticated probability estimate for unknown words, accuracy is 70.4% compared to 77.5% accuracy using the transformation-based approach. Next, we extended the suffix list to 23 suffixes and reran the experiment, obtaining an accuracy for unknown words of 71.7%.[11] Note that the transformation-based approach obtains a higher accuracy even though contextual information is used in the probabilistic approach and was not used in the transformation-based approach, and the resulting transformation-based annotator is completely symbolic. Also, no language-specific information is built into the transformation-learner. In the statistical approach, some of the assumptions include: a hyphen will be a good tagging cue, proper nouns are indicated by a combination of capitalization and position in a sentence, suffixes are good indicators of word class. In the transformation-based approach, none of this information is prespecified, but it is all learned. In the next section, we will improve upon both unknown word accuracy and known word accuracy by using context-triggered transformations.

**Brown Corpus**

The next experiment we ran used the Penn Treebank Tagged Brown Corpus. Sentences in the corpus were first randomly shuffled. Then the first 1,000 sentences (21,989 words) were used for the lexical training set, the second 1,000 sentences (23,050 words) were used for the contextual training set, the next 2,000 sentences (44,049 words) were used for testing and the entire unannotated corpus was used for the unannotated training corpus. 18.7% of word tokens in the test corpus do not appear in the lexical training corpus. For known words in the test corpus, tagging them with their most likely tag as indicated in the lexical training corpus results in an accuracy of 93.3%. Initially tagging all unknown words as singular common nouns results in an unknown word accuracy of 28.3%. 207

[11] The difference in accuracy using the probabilistic method quoted in [84] (85%) and the accuracy obtained in our implementation is due to the much smaller training set used to train our implementation.

transformations are learned, whose application to the test set results in an unknown word accuracy of 74.7%. A list of the first 20 learned transformations is shown in figure 6.11. Once again, changing the learning score threshold from 2 to 3 results in higher accuracy with fewer transformations. In particular, doing so results in 118 transformations and an accuracy of 74.9%. Next, we ran the stochastic tagger with stochastic unknown word recognition, using the extended suffix list. This resulted in an unknown word accuracy of 71.6%, lower than the accuracy obtained by the transformation-based learner despite the fact that contextual token information was not used by the transformation-based learner.

A few differences between the transformations learned on the Wall Street Journal and those learned on the Brown Corpus are worth noting. First, the transformation indicating that a word to the right of a dollar sign is likely a number is a much more useful transformation in the Wall Street Journal (transformation number 4) than in the Brown Corpus (transformation number 19). Probably due to the fact that business tends to be male-dominated, the transformation found using the Brown Corpus that states that a word that can appear to the right of the word *She* is a past tense verb is not learned when trained on the Wall Street Journal.

We next ran the same experiment on the Brown Corpus using the original Brown Corpus part of speech tags [44].[12] The original Brown Corpus tag set is considerably larger than the Penn Treebank tag set. In the Brown corpus, 65 Penn Treebank tags occur more than once, whereas 159 original Brown Corpus tags occur more than once. Once again, the corpus was randomly shuffled and was divided into an annotated lexical training corpus of 1000 sentences (21,731 words), an annotated contextual training corpus of 1000 sentences (21,008 words), and a test corpus of 2000 sentences (41,833 words). 19.1% of the tokens in the test corpus do not occur in the lexical training corpus. 248 lexical transformations were learned for tagging unknown words and the resulting accuracy on unknown words was 71.0%. Accuracy on known words was 92.5%. Changing the transformation-finding threshold from two to three also results in an unknown word accuracy of 71.0%, but with only 141 transformations. The fact that accuracy is greater when using the Penn Treebank tags is probably due to the fact that there are fewer such tags to choose from. In figure

[12] See appendix 3 for a description of this tag set.

|  | Change Tag |  |  |
|---|---|---|---|
| # | From | To | Condition |
| 1 | ?? | NNS | Suffix is *s* |
| 2 | ?? | NP | Can appear at the start of a sent. |
| 3 | ?? | VBN | Suffix is *ed* |
| 4 | ?? | JJ | Adding the suffix *ly* results in a word |
| 5 | ?? | RB | Suffix is *ly* |
| 6 | ?? | VBG | Suffix is *ing* |
| 7 | NN | VB | Can appear to the right of *could* |
| 8 | ?? | VBD | Can appear to the right of *She* |
| 9 | ?? | JJ | Character - appears in the word |
| 10 | ?? | CD | Character *1* appears in the word |
| 11 | ?? | NP | Character *C* appears in the word |
| 12 | ?? | NP | Character *S* appears in the word |
| 13 | NN | JJ | Suffix is *ic* |
| 14 | NN | JJ | Can appear to the right of *were* |
| 15 | ?? | NP | Character *P* appears in the word |
| 16 | NNS | NN | Suffix is *ss* |
| 17 | ?? | NP | Character *M* appears in the word |
| 18 | ?? | NP | Character *B* appears in the word |
| 19 | ?? | CD | Can appear to the right of *$* |
| 20 | JJ | NN | Can appear to the left of *'s* |

Figure 6.11: The first 20 transformations from the Penn Treebank Brown Corpus.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | ?? | nns | Suffix is *s* |
| 2 | ?? | vbn | Suffix is *ed* |
| 3 | ?? | jj | Adding the suffix *ly* results in a word |
| 4 | ?? | vbg | Suffix is *ing* |
| 5 | ?? | ly | Suffix is *ly* |
| 6 | nn | np | Can appear at the start of a sent. |
| 7 | ?? | vb | Can appear to the right of *can* |
| 8 | nns | np$ | Character *'* appears in the word |
| 9 | ?? | vbd | Can appear to the right of *She* |
| 10 | ?? | jj | Suffix is *ble* |
| 11 | ?? | cd | Character *1* appears in the word |
| 12 | ?? | jj | Suffix is *ic* |
| 13 | np | nn | Can appear to the left of *of* |
| 14 | ?? | np | Can appear to the right of *Mr.* |
| 15 | nns | nn | Suffix is *ss* |
| 16 | ?? | jj | Suffix is *al* |
| 17 | np$ | nn$ | Can appear to the right of *the* |
| 18 | nn | jj | Can appear to the right of *were* |
| 19 | jj | nn | Adding the suffix *'s* results in a word |
| 20 | nn | np | Prefix is *S* |

Figure 6.12: The first 20 transformations from the Original Brown Corpus.

6.12 we show the first 20 transformations that were learned. The eighth transformation, change a tag from plural common noun to possessive proper noun if an apostrophe appears in the word, appears when training using the original Brown Corpus tags, but not the Penn Treebank tags. This is because the Penn Treebank tokenizes words such as *America's* as *America 's*. We ran the stochastic tagger on the same training and test sets (using the large suffix list), obtaining an unknown word accuracy of 65.1%, again lower than that obtained by the transformation-based approach.

### Old English

As a step toward better understanding how general the learner really is, we next tested it on a corpus of Old English. The corpus used was the Helsinki Corpus of English Texts: Diachronic and Dialectal, which contains a varied collection of written Old English. We had originally planned to use Middle English, but the more rigid spelling conventions of Old English made this language more amenable to our learning algorithm. Old English is similar in many ways to modern Germanic languages such as German.[13] and is quite different from modern English. Old English has five morphological cases marked by endings on article, adjective and noun. Articles, adjectives and nouns are also marked for gender. Old English has a much freer word order than modern English. Generally, the verb occurs in final position in subordinate clauses, and in second position in main clauses. Complements and adjuncts do not have as rigid an order as in modern English.

We had access to a 500,000 word unannotated corpus, 25,831 words of which had been manually tagged.[14] As was done above, the annotated corpus was divided into three sub-corpora: one for lexical training, one for contextual training and one for testing. Because the annotated corpus was small, we allowed the two training sets to overlap. The training corpus contained 22,039 words and was divided into a lexical training corpus of 14,847 words and a contextual training corpus of 14,871 words. The test corpus contained 3,792 words. 20.6% of the word tokens in the test set did not occur in the lexical training set.

In training the lexical part of speech tagging module, 248 transformations were learned. Figure 6.13 shows the first 20 learned transformations. Initially, all unknown words are tagged as singular common nouns (NN). A graph of accuracy as a function of transformation number on the test corpus can be seen in figure 6.14. The reason for the unusual shape of this curve, compared to the much smoother curves obtained on the English corpora, is a bit of a mystery. Initial accuracy obtained by tagging all unknown words as nouns is 43.7%. One reason this is significantly higher than in the English corpora is that the smaller tag set used for Old English did not differentiate between proper nouns and common nouns, nor did it distinguish between singular and plural nouns. The tag set used for Old English can be found in appendix 2. After applying the transformations, an accuracy of 67.2% was obtained. When the transformations were pruned by discarding those that resulted in a decrease in accuracy when applied to the contextual training corpus, this reduced the transformation list to 218 transformations, whose application to the test set resulted in a slightly higher accuracy of 68.0%. If the threshold on transformation scores is raised from

[13] Thanks to Eric Haeberli for providing the linguistic details on Old English.
[14] The manual tagging was done by Eric Haeberli.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | ?? | VT | Can appear to the right of *ne* |
| 2 | ?? | VT | Suffix is *+d* |
| 3 | ?? | VT | Suffix is *on* |
| 4 | ?? | NN | Can appear to the right of *his* |
| 5 | ?? | VN | Suffix is *an* |
| 6 | VN | NN | Deleting the suffix *n* results in a word |
| 7 | ?? | RB | Suffix is *lice* |
| 8 | ?? | VT | Suffix is *ode* |
| 9 | ?? | VT | Suffix is *+t* |
| 10 | ?? | VN | Suffix is *nne* |
| 11 | ?? | VT | Can appear to the right of *He* |
| 12 | NN | JJ | Can appear to the right of *swi+de* |
| 13 | ?? | VBN | Suffix is *ed* |
| 14 | ?? | VBG | Suffix is *ende* |
| 15 | ?? | NN | Can appear to the right of *for* |
| 16 | NN | VT | Deleting the prefix *a* results in a word |
| 17 | NN | VT | Deleting the prefix *for* results in a word |
| 18 | ?? | NN | Can appear to the right of *+t+are* |
| 19 | NN | VT | Prefix is *on* |
| 20 | ?? | VT | Suffix is *st* |

Figure 6.13: The first 20 transformations from the Old English Corpus.

2 to 3 on the unpruned list of transformations, an accuracy of 67.1% is obtained on only 120 transformations.

In table 6.2, we summarize the results of this section, obtained prior to using any contextual token information. Note that these experiments were all done on small training corpora, since portability is an important issue. As it has been shown that taggers trained on one domain do not tag with high accuracy when tested on a different domain ([84]), we do not think that a tagger requiring millions of words of tagged text for training is very practical in many real-world situations where a part of speech tagger is desired. Later in this section we will present results for larger training corpora. Known words are tagged with their most likely tag as indicated in the lexical training corpus. Unknown words are tagged by first assigning them a default most likely tag, and then applying the learned transformations. Notice that even prior to incorporating context-triggered transformations,

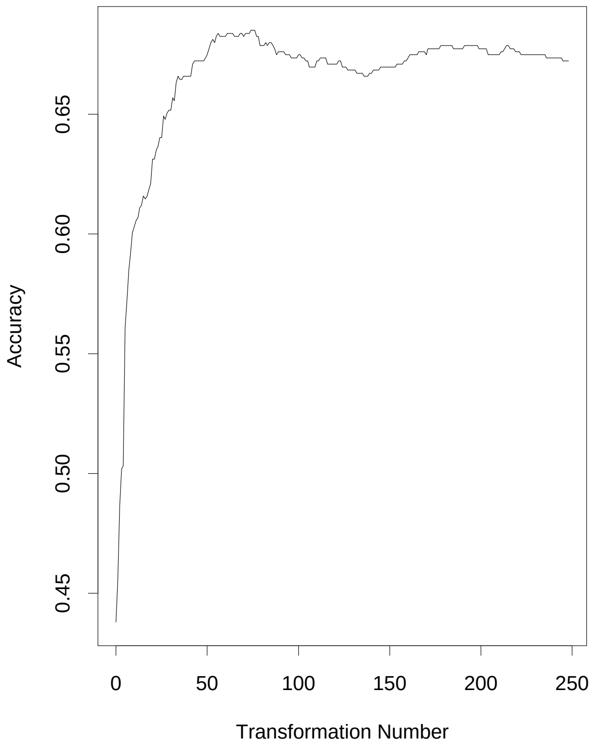



Figure 6.14: Unknown Word Tagging Accuracy After Applying Transformations.

| Method | Corpus | Unknown Words | Known Words | Total |
|---|---|---|---|---|
| Lexical Transformations | WSJ | 77.5 | 93.6 | 90.5 |
| Probabilistic Tagging | WSJ | 71.7 | 95.4 | 91.0 |
| Lexical Transformations | Brown - Penn Tags | 74.7 | 93.3 | 89.9 |
| Probabilistic Tagging | Brown - Penn Tags | 71.6 | 94.7 | 90.3 |
| Lexical Transformations | Brown - Orig Tags | 71.0 | 92.5 | 88.4 |
| Probabilistic Tagging | Brown - Orig Tags | 65.1 | 94.8 | 89.1 |
| Lexical Transformations | Old English | 67.2 | 88.6 | 84.2 |

Table 6.2: Summary of Accuracy of Lexical Learning.

the transformation-based approach significantly outperforms the probabilistic approach at tagging unknown words, at least for relatively small training corpora. We could not implement the stochastic tagger to run on the Old English corpus because it is not truly portable like the transformation-based tagger is; a list of significant affixes in Old English, as well as knowledge of other tagging cues, would have been necessary to run the stochastic tagger on this corpus.

## 6.3 Learning Context Triggered Transformations to Improve Accuracy

After learning the most likely tag for words appearing in the small annotated lexical training corpus as well as a method of predicting the most likely tag for unfamiliar words, the next step is to use contextual cues to disambiguate word tokens. To do this, we once again use a transformation-based learner. In [16], we describe a transformation-based part of speech tagger.[15] This tagger works by first tagging every word with its most probable part of speech estimated from a large corpus of annotated text, and then automatically learning a small set of contextually triggered transformations to improve tagging performance. We

[15] What we are doing is similar to decision tree learning [91], but with a decision tree sparse data problems abound, since (in a binary decision tree) every level deeper in the tree has only half the training material (on the average) of the next level up. Yet, we have some evidence that at least in the domain of tagging, the two methods have comparable error rates (see [7]).

demonstrated that with fewer than 100 such symbolic contextual transformations, performance was obtained that was comparable to stochastic taggers that capture contextual information in tens of thousands of contextual probabilities and use smoothing techniques for overcoming the problem of estimated probabilities of zero. In particular, tagging accuracy was 95% without an external dictionary when trained on 90% of the Brown Corpus and tested on a held-out test set, and was 96% when using a dictionary derived from the entire Brown Corpus. To give an example of how the transformation-based system can capture contextual information so much more concisely, look at transformation number 2 in figure 6.15, which says that a tag should be changed from *VBP* to *VB* if one of the previous three tags is a modal. To express this in terms of contextual probabilities, we would need statistics on all 4-grams of the form:

- MD * * VB
- MD * * VBP
- * MD * VB
- * MD * VBP
- * * MD VB
- * * MD VBP

where * can be any part of speech tag. This large set of statistics, along with a method of smoothing to handle those 4-grams not occurring in the training corpus, would capture information comparable to the single transformation through the relative counts of different 4-grams ending in *VB* and *VBP*.

Although only a small annotated corpus was needed for learning context-triggered transformations in [16], a very large annotated corpus (over one million words) was used for training the most-likely-tag tagger as well as the procedure for tagging unknown words. In addition, this tagger included one manually created rule for distinguishing between common nouns and proper nouns. The tagger had the same weakness as other taggers in not being very portable.

The work described here is different from the earlier transformation-based tagger in two ways. First, it uses most likely tag information learned in the module discussed in the last section, thereby requiring significantly less human labor in preparing training material. Second, there are no specific assumptions built into the learner/tagger. There are built-in assumptions about the types of cues we anticipate as reflected by the set of transformation templates. But, all specific information, such as cues to be used for distinguishing between proper and common nouns, is learned. This makes the system much more portable. In addition, in the transformation-based tagger described in [16], the tagging of a word in the test set that was also seen in the training set is only changed to X if the word was tagged with X somewhere in the training corpus. Because the training corpus we use here is much smaller, we can not assume that any sort of closure is reached in the training corpus on the set of allowable tags for words.

The following templates are used in the context-triggered transformation learner:[16]

- Change a tag from X to Y if:
  - The previous (following) word is tagged with Z.
  - The previous word is tagged with Z and the following word is tagged with W.
  - The following (preceding) two words are tagged with Z and W.
  - One of the two preceding (following) words is tagged with Z.
  - One of the three preceding (following) words is tagged with Z.
  - The word two words before (after) is tagged with Z.

Unlike the transformation-based module for lexical information, the contextual information module uses a score based on per token performance instead of per type performance. This is because the contextual module discovers transformations to be applied to word tokens in particular environments, whereas the lexical module discovers transformations

[16] This transformation list could easily be extended to make reference to words, word classes, and different properties of words. If transforamtions with word triggering environments are added, then this method has an advantage over stochastic taggers in that relationships between two words or between a word and the tag of another word can explicitly be captured within the transformation-based framework (change the tag from X to Y if the previous word is Z), whereas this cannot be done in the current statistical approach where words are first mapped to tags, and then tag sequence information is used without making reference to the underlying words.

to be applied to word types regardless of context. Note the way bigram information is used here, compared to its use in the first stage of tagging. In the first stage, where most likely tag information is learned, transformations involving bigrams state that a tag should be changed if a word is *ever* seen next to a particular word. In this phase of learning, a tag is changed only if a particular instance of a word *is* next to a particular word. The score for a transformation is simply the per token tagging accuracy resulting from applying the transformation. The score for a transformation from tag X to tag Y is measured as the number of times this transformation applies to a word that should be tagged as Y but is currently tagged as X (positive change) minus the number of times the transformation applies to a word that is currently and correctly tagged as X (negative change). Once again, a greedy search is carried out to discover a set of transformations, with the best scoring transformation being added to the transformation list at every learning iteration. Training run time is the same as for the unknown word learning module: $O(|op| * |env| * |n|)$, where $|op|$ is the number of allowable transformation operations (change tag from X to Y, for all tags X,Y), $|env|$ is the number of possible triggering environments, and $|n|$ is the training corpus size. In this module, the training corpus size is the number of word tokens, and not word types. Applying contextual transformations takes $O(|T| * |n|)$ time, where $|T|$ is the size of the transformation list and $|n|$ is the number of word tokens to be tagged.

### 6.3.1 Results

#### Wall Street Journal

The contextual training corpus was used for learning contextual transformations. This corpus is first tagged using the lexical start-state described in the previous section. This annotator has two parts: a listing of words and their most likely tag for words seen in the annotated lexical training corpus, and a procedure (list of transformations) for tagging words not occurring in this corpus. This lexical information is used to initially tag the contextual training corpus. Next, the contextual transformation learner is run on this corpus. A list of the first twenty contextual transformations that were learned can be found in table 6.15. In total, 175 transformations were learned. In table 6.3, we show tagging accuracy both before and after applying contextual transformations and compare these results to results obtained using the probabilistic approach of [84] which was described above. The transformation-based approach performs slightly worse than the statistical tagger on known words (.1% at 1,000 sentences lexical and contextual training corpora, .6% at 4,000 sentences), but does significantly better on unknown words and overall.[17] As the percentage of unknown words decreases as training corpus increases, it is likely that the statistical approach will somewhat outperform the transformation-based approach when using much larger corpora. For instance, we reran the experiments on 4,000 lexical and contextual training corpora after adding a lexicon built from an additional 50,000 sentences.[18] The results from this experiment are shown in 6.4. In this experiment, the transformation-based tagger performed slightly worse than the stochastic tagger. We hope that extending the transformation list in the future will lead to an improvement in performance on known words. With the current set of transformations, it appears that the transformation-based system significantly outperforms the stochastic tagger on a small corpus (49,000 words), obtains slightly better performance when trained on 200,000 words, and obtains slightly worse performance when using around a million words of training material. The transformation-based tagger contains absolutely no prespecified language-specific or corpus-specific information, and relies on no external aids such as dictionaries or affix lists.

Two other results are quoted in the literature for stochastic taggers trained on much larger training samples of the Penn Treebank. In [84], an accuracy of 96.3% is obtained when training on one million words. However, the lexicon was closed in the sense that it was built from both the training and test set, and so there were no unknown words. In [7] an accuracy of 95.4% was obtained training on over 4.4 million words. We have achieved results that are competitive with results quoted in the literature for stochastic taggers trained on large corpora. This is significant given that the tagger itself is completely symbolic, and is able to capture information to be used in tagging much more concisely. In stochastic tagging, the lexicon contains entries of the form:

[17] When training on larger corpora, the transformation-based tagger could probably be improved significantly by constraining rules to change a tag from X to Y only if the word has been tagged with Y somewhere in the training corpus.

[18] These sentences did not include the test set.

| | Unknown Word Accuracy | Known Word Accuracy | Total |
|---|---|---|---|
| 1,000 Sentence Lexical and Contextual Training Corpora | | | |
| Lexical Transformations | 77.5 | 93.6 | 90.5 |
| Lexical and Contextual Transformations | 81.2 | 95.3 | 92.7 |
| Probabilistic (Small Suffix List) | 70.4 | 95.4 | 90.7 |
| Probabilistic (Big Suffix List) | 71.7 | 95.4 | 91.0 |
| 4,000 Sentence Lexical and Contextual Training Corpora | | | |
| Lexical Transformations | 81.3 | 93.4 | 92.2 |
| Lexical and Contextual Transformations | 84.3 | 95.5 | 94.4 |
| Probabilistic (Big Suffix List) | 75.2 | 96.1 | 94.1 |

Table 6.3: Wall Street Journal Tagging Results.

**WORD TAG** $P(WORD|TAG)$

for all word-tag pairs seen in the training corpus. The transformation-based lexicon contains only one lexical entry for each word, of the form:

**WORD TAG**

indicating the most common tag for a word in the training corpus. Contextual information is also more concise: in one of the Wall Street Journal tagging experiments, contextual information was expressed in 7,731 trigram probabilities in the stochastic tagger, and was expressed in 206 transformations in the transformation-based tagger.

It should be noted that the transformation-based tagger currently has an environment window of three words, while a trigram tagger has an environment of only two words. To extend the trigram to a 4-gram tagger in order to use a larger context would result in an exponential increase in the number of contextual probabilities, as well as an exponential increase in the run-time of the tagger.

In table 6.5, the top twenty word tagging errors are shown that result after both lexical and contextual transformations are applied. The first error is a result of the fact that a double dash was not encountered in the annotated lexical training corpus, but did appear in the test corpus. An example of the second most frequent lexical error appears in: *The result*

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | NN | VB | Previous tag is *TO* |
| 2 | VBP | VB | One of the previous 3 tags is *MD* |
| 3 | NN | VB | Previous tag is *MD* |
| 4 | VBD | VBN | One of the previous 2 tags is *VBP* |
| 5 | VBN | VBD | Previous tag is *NP* |
| 6 | VBD | VBN | One of the previous 2 tags is *VBZ* |
| 7 | VBN | VBD | Previous tag is *PP* |
| 8 | POS | VBZ | Previous tag is *PP* |
| 9 | VB | VBP | Previous tag is *NNS* |
| 10 | VBP | VB | Previous tag is *TO* |
| 11 | VB | VBP | Previous tag is *PP* |
| 12 | JJ | NN | The surrounding tags are *DT* and *IN* |
| 13 | VBD | VBN | Previous tag is *VBD* |
| 14 | VB | NN | One of the previous two tags is *DT* |
| 15 | IN | WDT | The surrounding tags are *NN* and *VBZ* |
| 16 | NN | VBP | Previous tag is *PP* |
| 17 | NP | NN | The surrounding tags are *START* and *NNS* |
| 18 | NNS | NP | Following tag is *NP* |
| 19 | RBR | JJR | One of the following 3 tags is *NNS* |
| 20 | VBD | VBN | One of the previous 2 tags is *VB* |

Figure 6.15: The first 20 contextual transformations from the WSJ.

| | Unknown Word Accuracy | Known Word Accuracy | Total |
|---|---|---|---|
| Lexical and Contextual Transformations | 83.4 | 95.7 | 95.3 |
| Probabilistic | 76.7 | 96.3 | 95.7 |

Table 6.4: Wall Street Journal Tagging Results: Using A Much Larger Lexicon (Adding 50,000 Sentences).

*not only produces government of dubious legitimacy but consequences* **that** *often have been perverse .* In figure 6.16, the top twenty tag confusions are shown. The confusion between adjectives and nouns accounts for 15.6% of the total error. This is in part due to the difficulty human annotators have in choosing the appropriate tag for words in compound nouns such as *American Indians.* Below we show twenty randomly chosen sentences from the test corpus. Tagging errors are shown highlighted in the form *word/system-tag/correct tag.*

1. Miller/NP Brewing/NP Co./NP has/VBZ filed/VBN suit/NN against/IN a/DT physicians/NNS group/NN **that/IN/WDT sold/VBN/VBD**
   T-shirts/NNS mocking/VBG
   its/PP$ Miller/NP Lite/NP ad/NN campaign/NN in/IN Texas/NP ./.

2. And/CC those/DT snags/NNS have/VBP emerged/VBN less/JJR than/IN six/CD months/NNS after/IN the/DT departure/NN of/IN one/CD of/IN its/PP$ **founding/VBG/NN** partners/NNS ,/, Leonard/NP Green/NP ./.

3. Such/JJ activities/NNS do/VBP n't/RB **appear/VBP/VB** to/TO meet/VB the/DT standard/NN set/VBN in/IN the/DT endowment/NN 's/POS "/" statement/NN of/IN principles/NNS and/CC objectives/NNS ,/, "/" which/WDT **mandated/VBN/VBD** that/IN the/DT organization/NN "/" will/MD not/RB pick/VB and/CC choose/VB among/IN the/DT democratic/JJ competitors/NNS in/IN countries/NNS where/WRB such/JJ competition/NN is/VBZ possible/JJ ./. "/"

4. In/IN the/DT derivative/JJ market/NN ,/, new-issue/JJ activity/NN slowed/VBD after/IN a/DT busy/JJ session/NN Wednesday/NP ,/, when/WRB five/CD real/JJ estate/NN mortgage/NN investment/NN conduits/NNS totaling/VBG $/$ 2/CD billion/CD were/VBD priced/VBN ./.

5. And/CC you/PP can/MD not/RB **put/VBN/VB in/IN/RB enough/RB/JJ Federal/NP/JJ** money/NN ,/, even/RB if/IN we/PP had/VBD it/PP ,/, to/TO solve/VB the/DT problem/NN ./. "/"

6. As/IN a/DT/DT **result/VB/NN** ,/, I/PP 'm/VBP especially/RB keen/JJ on/IN providing/VBG **wheelchair/WRB/NN** ramps/NNS ,/, lifts/NNS ,/, signers/NNS for/IN **deaf/NN/JJ** people/NNS ,/, and/CC readers/NNS for/IN the/DT **blind/JJ/NN** ./.

7. Mr./NP/NP Spielvogel/NP/NP repeated/JJ/VBD his/PP/*PP* offer/NN/NN yesterday/NN/NN ,/,/, he/PP/PP said/VBD/VBD ,/,/, when/WRB/WRB
   Mr./NP/NP Saatchi/NP/NP called/VBN/VBD to/TO/TO tell/VB/VB him/PP/PP about/IN/IN the/DT/DT management/NN/NN restructuring/NN/NN ././.

8. **Thus/DT/RB** Mr./NP LeBow/NP has/VBZ n't/RB been/VBN able/JJ to/TO achieve/VB the/DT hoped-for/JJ **cash-flow/JJ/NN** boost/NN ,/, despite/IN replacing/VBG 75/CD %/NN of/IN management/NN ,/, firing/VBG and/CC retiring/VBG 2,800/CD employees/NNS ,/, selling/VBG most/JJS of/IN **Western/JJ/NP Union/NP/NP** 's/POS **telecommunications/NN/NNS** assets/NNS and/CC cutting/VBG $/$ 134/CD million/CD in/IN annual/JJ **operating/VBG/NN** costs/NNS ./.

9. Mr./NP Spielvogel/NP **replied/VBN/VBD** that/IN he/PP was/VBD n't/RB **interested/JJ/VBN** in/IN a/DT **hostile/NN/JJ** attempt/NN ./.

10. The/DT main/JJ reason/NN for/IN the/DT **split/JJ/NN** was/VBD "/" to/TO make/VB the/DT stock/NN more/RBR attractive/JJ to/TO the/DT **retail/NN/JJ** buyer/NN ,/, "/" says/VBZ Elliott/NP J./NP Horowitz/NP ,/, the/DT company/NN 's/POS **executive/NN/JJ** vice/NN president/NN and/CC chief/NN financial/JJ officer/NN ./.

11. But/CC his/PP$ joint/JJ appearance/NN with/IN the/DT speaker/NN appeared/VBD to/TO be/VB part/NN of/IN an/DT effort/NN to/TO **harness/NN/VB** what/WP momentum/NN the/DT party/NN can/MD capture/VB from/IN the/DT **resurgent/NN/JJ** power/NN of/IN the/DT pro-choice/JJ movement/NN ./.

12. Indeed/RB ,/, Mr./NP Roh/NP said/VBD tensions/NNS in/IN Asia/NP are/VBP rising/VBG ,/, not/RB falling/VBG ./.

13. Until/IN the/DT carrier/NN 's/POS new/JJ owner/NN ,/, **Alfred/VBN/NP** Checchi/NP took/VBD control/NN of/IN NWA/NP in/IN August/NP following/VBG a/DT $/$ 3.65/CD billion/CD buy-out/NN ,/, many/JJ industry/NN observers/NNS **figured/VBN/VBD** that/IN Europe/NP 's/POS **Airbus/JJ/NP** Industrie/NP had/VBD the/DT **inside/NN/JJ** track/NN on/IN **winning/JJ/VBG** plane/NN orders/NNS from/IN NWA/NP ./.

14. **Pleasant/NP/JJ** neighborhoods/NNS **sit/VB/VBP** on/IN rolling/VBG hills/NNS ./.

15. **Yet/CC/RB** we/PP 've/VBP had/VBD 100/CD years/NNS of/IN poverty/NN ,/, "/" says/VBZ Ed/NP Jones/NP ,/, a/DT retired/VBN Tennessee/NP congressman/NN and/CC a/DT member/NN of/IN the/DT commission/NN ./.

16. **Were/NP/VBD** I/PP to/TO stay/VB ,/, there/EX would/MD n't/RB be/VB very/RB **much/RB/JJ** to/TO **do/VBP/VB** ./. "/"

17. Initially/RB ,/, Sony/NP had/VBD said/VBD it/PP would/MD n't/RB complete/VB the/DT purchase/NN of/IN Guber/NP Peters/NP unless/IN the/DT matter/NN was/VBD **resolved/VBD/VBN** ./.

18. **Clearly/NP/RB** ,/, the/DT company/NN is/VBZ going/VBG to/TO **show/NN/VB** improved/VBN **operating/VBG/NN** profit/NN ,/, having/VBG **delivered/VBD/VBN** an/DT impressive/JJ 86/CD jets/NNS during/IN the/DT period/NN ./.

19. Its/PP$ current/JJ predicament/NN suggests/VBZ how/WRB even/RB the/DT sharpest/JJS financial/JJ minds/NNS **/JJ/:** those/DT of/IN Mr./NP LeBow/NP and/CC his/PP$ Drexel/NP partners/NNS **–/JJ/:** can/MD **stumble/JJ/VB** in/IN the/DT **face/VB/NN** of/IN a/DT revolution/NN in/IN technology/NN ./.

| Tagged As | Should Be | Word | % of Total Error | Occurred in Training Set |
|---|---|---|---|---|
| JJ | : | – | 3.13 | no |
| IN | WDT | that | 1.66 | yes |
| IN | RB | about | 1.31 | yes |
| JJR | RBR | earlier | .86 | yes |
| RB | JJ | much | .74 | yes |
| IN | DT | that | .72 | yes |
| IN | RB | up | .65 | yes |
| JJ | NN | virus | .63 | no |
| NN | JJ | chief | .61 | yes |
| JJR | RBR | more | .57 | yes |
| JJ | NP | Western | .55 | yes |
| IN | RB | as | .53 | yes |
| NN | JJ | executive | .47 | yes |
| RBR | JJR | more | .45 | yes |
| VBG | NN | operating | .41 | yes |
| POS | VBZ | 's | .41 | yes |
| NN | RB | back | .41 | yes |
| IN | RB | ago | .41 | yes |
| NNS | NN | basis | .39 | no |
| JJS | RBS | most | .39 | yes |

Table 6.5: The Top Twenty Word Tagging Errors: Wall Street Journal.

20. In/IN **contrast/VBP/NN** ,/, the/DT **Big/JJ/NP** Board/NP 's/POS number/NN of/IN companies/NNS remained/VBD at/IN a/DT steady/JJ 1,681/CD ,/, and/CC the/DT Nasdaq/NP 's/POS roster/NN **ballooned/VBN/VBD** to/TO 4,451/CD ./.

### Brown Corpus

In table 6.6, we show the results from learning both lexical and contextual transformations for the Brown Corpus. Results were obtained using both the Penn Treebank part of speech tags and the original Brown Corpus tags. Using the Penn Treebank tags, 160 contextual transformations were learned. The first 20 contextual transformations are shown in figure 6.17. Using the Brown Corpus tags, 209 transformations were learned. The first 20 transformations learned using Brown Corpus tags are shown in figure 6.18. Using both tag

| Tagged As | Should Be | % of Total Error |
|---|---|---|
| NN | JJ | 9.80 |
| JJ | NN | 5.81 |
| VBN | VBD | 5.75 |
| IN | RB | 3.76 |
| NNS | VBZ | 3.13 |
| JJ | : | 3.13 |
| JJ | NP | 3.01 |
| VBD | VBN | 2.66 |
| VBG | NN | 2.39 |
| NN | VBG | 2.33 |
| NP | JJ | 2.19 |
| JJR | RBR | 2.05 |
| NN | NP | 2.02 |
| JJ | VBN | 2.02 |
| NN | VBP | 1.94 |
| VBZ | NNS | 1.72 |
| NN | RB | 1.72 |
| NP | NN | 1.68 |
| RB | JJ | 1.66 |
| IN | WDT | 1.66 |

Figure 6.16: The Top Twenty Tagging Errors: Wall Street Journal.

| | Tag Set | Unknown Word Accuracy | Known Word Accuracy | Total |
|---|---|---|---|---|
| Lexical Transformations | Penn | 74.7 | 93.3 | 89.9 |
| Lexical and Contextual | Penn | 80.8 | 94.5 | 91.8 |
| Statistical Tagging | Penn | 71.6 | 94.7 | 90.3 |
| Lexical Transformations | Brown | 71.0 | 92.5 | 88.4 |
| Lexical and Contextual | Brown | 75.0 | 94.6 | 90.9 |
| Statistical Tagging | Brown | 65.1 | 94.8 | 89.1 |

Table 6.6: Tagging The Brown Corpus.

sets, the transformation based approach significantly outperforms the statistical approach on unknown words and on the corpus as a whole, and only performs marginally worse on known words.

### Old English

When training on Old English, 81 contextual transformations were learned, the first 10 of which are shown in figure 6.19. A table showing the resulting accuracy is seen in figure 6.20. Accuracy is somewhat lower than the accuracy obtained on the English corpora. There are a number of reasons for this. A smaller corpus (about half the size of the training corpora used in the other lexical learning experiments) was used and the two annotated training corpora overlapped. Overlapping is a problem because this means that the contextual transformation learner is learning on a corpus that is unlike the corpus the learned transformations are applied to; since the corpora overlap, there will be fewer unknown words in the contextual training corpus than if they had not overlapped. Since Old English has a much freer word order, an order-independent transformation such as triggered by tag X appearing a certain number of words on *either* side of a particular word might prove to be more effective. Also, the Old English corpus had more typographical errors from manual annotation than are found in the English corpora. Despite these problems, we are encouraged that this level of accuracy was obtained on a language other than English.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | NN | VB | Previous tag is TO |
| 2 | VBN | VBD | Previous tag is PP |
| 3 | VB | VBP | Previous tag is PP |
| 4 | NN | VB | Previous tag is MD |
| 5 | VBP | VB | One of the previous 3 tags is MD |
| 6 | VBN | VBD | Previous tag is NP |
| 7 | VB | NN | Previous tag is DT |
| 8 | VBD | VBN | Previous tag is VBD |
| 9 | VBD | VBN | One of the previous 2 tags is VB |
| 10 | VBP | VB | One of the previous 3 tags is TO |
| 11 | VBD | VBN | Previous tag is VBZ |
| 12 | VB | VBP | Previous tag is NNS |
| 13 | POS | VBZ | One of the previous 3 tags is PP |
| 14 | NNS | VBZ | Next tag is DT |
| 15 | NN | NBP | Previous tag is PP |
| 16 | VBG | NN | The surrounding tags are DT and IN |
| 17 | RBR | JJR | Next tag is IN |
| 18 | RB | JJ | The surrounding tags are DT and NN |
| 19 | VB | NN | Previous tag is JJ |
| 20 | VB | NN | Previous tag is NN |

Figure 6.17: Contextually Triggered Transformations For the Brown Corpus Using Penn Treebank Tags.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | to | in | Next tag is *at* |
| 2 | vbn | vbd | One of the previous 3 tags is *Start of Sent* |
| 3 | vb | nn | One of the previous 2 tags is *at* |
| 4 | nn | vb | Previous tag is *to* |
| 5 | vbd | vbn | One of the previous 3 tags is *hvd* |
| 6 | nn | vb | One of the previous 2 tags is *md* |
| 7 | vbn | vbd | Previous tag is *np* |
| 8 | to | in | Next tag is *np* |
| 9 | vbn | vbd | Previous tag is *pps* |
| 10 | to | in | One of next 2 tags is *nns* |
| 11 | vbd | vbn | One of the previous 3 tags is *be* |
| 12 | to | in | Next tag is *pp$* |
| 13 | vbn | vbd | Previous tag is *ppss* |
| 14 | vbd | vbn | One of previous 3 tags is *hvz* |
| 15 | ppss | ppo | Previous tag is *vb* |
| 16 | vbd | vbn | One of previous 2 tags is *hv* |
| 17 | vbd | vbn | One of previous 2 tags is *bedz* |
| 18 | vbd | vbn | One of previous 3 tags is *bez* |
| 19 | vb | nn | One of previous 2 tags is *in* |
| 20 | to | in | Next tag is *ppo* |

Figure 6.18: Contextually Triggered Transformations For the Brown Corpus Using Original Brown Corpus Tags.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | DT | NN | Next tag is *CO* |
| 2 | DT | RB | Next tag is *VT* |
| 3 | RB | PR | The surrounding tags are *RB* and *NP* |
| 4 | DT | RB | Next tag is *PR* |
| 5 | NE | CC | One of the following two tags is *NN* |
| 6 | PDT | DT | Next tag is *NN* |
| 7 | RB | PR | The surrounding tags are , and *NP* |
| 8 | DT | PR | Next tag is *NP* |
| 9 | CO | DT | The surrounding tags are *PR* and *NN* |
| 10 | CO | NN | The previous tag is *START* |

Figure 6.19: The first 10 contextual transformations from the Old English Corpus.

| | Unknown Word Accuracy | Known Word Accuracy | Total |
|---|---|---|---|
| Lexical Transformations | 67.2 | 88.6 | 84.2 |
| Lexical and Contextual Transformations | 68.8 | 90.3 | 85.9 |

Figure 6.20: Old English Tagging Results.

## 6.4 Conclusions

In this chapter, we have demonstrated that transformation-based error-driven learning can be used to effectively tag text. First, transformations are learned to guess the most likely tag for unknown words. Next, contextual transformations are used to tag words based upon the context they appear in. The transformation-based approach was shown to outperform the well-known statistical approach to tagging when training on small corpora, and to obtain comparable performance on larger corpora, despite the fact that information is stored much more compactly, no probabilities are used in tagging, and no smoothing of the probabilities of unobserved events is needed. In addition, no language specific or corpus specific knowledge is hardwired in the transformation-based tagger, thereby making it highly portable. The system can be simply extended by providing the learner with additional transformation templates. We demonstrated that with absolutely no changes to the program, it could be used to tag Old English simply by providing it with a small tagged corpus and larger untagged corpus as training material.

# Chapter 7

# Phrase Structure

We next turn our attention from word classes to phrase structure. A number of proposals came out of the American Structural linguistics school on how a field linguist could determine the phrase structure of sentences in an unfamiliar language [33, 61, 52, 53, 111]. We described these approaches in an earlier section. All of these approaches require a trained linguist working with an informant to tease out the structural information of a sentence. The field linguist is permitted to ask anything of an informant, in a sense giving him access to an infinite corpus from which linguistic information can be learned. We wanted to determine to what extent it was possible to learn phrase structure information from a finite (and preferably very small) sample corpus. Below we describe a module of the learning system that automatically learns phrase structure, given only a small corpus annotated with skeletal brackets and part of speech tags as input.[1] The learning module is able to assign a phrase structure analysis to sentences tagged with parts of speech with high accuracy.[2]

There have been several other recent proposals for automatic phrase structure learning based on statistics gathered over large corpora. In [106, 79], a statistic based on mutual information is used to find phrase boundaries. The key idea used in these papers is that a position between two words with relatively low mutual information between strings on

[1] Of course, the input sentences need not be manually tagged. They can be tagged using the minimal resource tagger described in the previous section, or any of the many other available taggers if sufficient training material is available.

[2] This work has also been reported in [17, 18, 19].

the left and strings on the right is likely to be a phrase boundary. [100] defines a function to score the quality of parse trees and a move set, and then uses simulated annealing to heuristically explore the entire space of possible parses for a given sentence. In [23], distributional analysis techniques are applied to a large corpus to learn a context-free grammar. Rules are of the form $a \rightarrow b\ c$, where a, b and c are all part of speech tags. The score for a rule is based on the distributional similarity, measured by taking the relative entropy of adjacent word distributions, for the single tag on the left hand side and the pair of tags on the right hand side of the rule. A rule such as *pronoun* $\rightarrow$ *determiner noun* would have a good score, since a pronoun and a noun phrase are distributionally similar.

None of these methods were tested in a way that allows them to be readily compared to other methods. In [12], statistics are calculated on all possible subtrees contained in a structurally annotated training corpus. A Monte Carlo technique [50] is then used to combine subtrees when parsing fresh text. This technique has been shown to be effective on a corpus from a very constrained domain, but it may not be possible to scale it up to effectively parse richer domains. In addition, as we will see below, this method performed worse than the transformation-based approach when trained on very small corpora used in our experiments.

The most promising results to date have been based on the inside-outside algorithm, which can be used to train stochastic context-free grammars. The inside-outside algorithm is an extension of the finite-state based Hidden Markov model (by [3]), which has been applied successfully in many areas, including speech recognition and part of speech tagging. A number of recent papers have explored the potential of using the inside-outside algorithm to automatically learn a grammar [75, 104, 88, 26, 31, 102]. In the inside-outside algorithm, context-free rule probabilities are incrementally altered in a way that increases the probability of the training corpus. The algorithm is guaranteed to converge upon a locally optimal set of rule probabilities with respect to training corpus probability, but is not guaranteed to find a globally optimal set. The inside-outside algorithm can be used to assign probabilities to a symbolic grammar written by a grammarian, or to learn a grammar automatically. In [88, 102], an initial grammar consisting of all possible binary rules (for a particular set of preterminals and nonterminals) is built, and each rule is assigned a random probability. The inside-outside algorithm is then applied to adjust these probabilities.

## 7.1 Building a Tree With Nonterminals Unlabelled

Below, we describe a new technique for grammar induction, using transformation-based error-driven learning. Unlike more common parsing techniques, nonterminals are added after the bracketing phase is completed, rather than being added concurrent with bracketing.[3] In addition to teasing apart the parsing problem into two simpler subproblems, this has the added advantage that the bracketing module can be trained on text without nonterminal information. The algorithm works by beginning in a very naive state of knowledge about phrase structure. By repeatedly comparing the results of parsing in the current state to the proper phrase structure for each sentence in the training corpus, the system learns a set of ordered transformations which can be applied to reduce parsing error. We believe this technique has advantages over other methods of phrase structure induction. Some of the advantages include: the system is very simple and can easily be extended, it requires only a very small set of transformations, a high degree of accuracy is achieved, and only a very small training corpus is necessary. The trained transformational parser is completely symbolic and can bracket text in linear time with respect to sentence length ($O(n)$ time, compared to $O(n^3)$ time for context-free grammar parsing). In addition, since some tokens in a sentence are not even considered in parsing, the method could prove to be considerably more robust than a CFG-based approach when faced with noise or unfamiliar input. After describing the algorithm, we present results and compare these results to other recent results in automatic phrase structure induction. Next, we present results obtained from training the system on a corpus annotated with part of speech tags using the lexical and contextual learning modules described in the previous section.

[3] In [77, 67], it is shown that nonterminals are not necessary in the sense that for every context free grammar there is a skeletal generating system (a set of trees without nonterminal labels and tree rewriting rules) that generates the same set of strings.

### 7.1.1 The Algorithm

The learning algorithm is trained on a small corpus of partially bracketed text which is also annotated with part of speech information.[4] The learner begins in a naive initial state, knowing very little about the phrase structure of the target corpus. In particular, all that is initially known is that English tends to be right branching and that final punctuation is final punctuation. Transformations are then learned automatically which transform the output of the naive parser into output which better resembles the phrase structure found in the training corpus. Once a set of transformations has been learned, the system is capable of taking sentences tagged with parts of speech (either manually tagged text, or the output of an automatic part of speech tagger) and returning a binary-branching structure with nonterminals unlabelled.[5]

#### The Initial State of the Parser

Initially, the parser operates by assigning a right-linear structure to all sentences. The only exception is that final punctuation is attached high. So, the sentence "*The dog and old cat ate .*" would be incorrectly bracketed as:

( ( The ( dog ( and ( old ( cat ate ) ) ) ) ) . )

The parser in its initial state will obviously not bracket sentences with great accuracy. In some experiments below, we begin with an even more naive initial state of knowledge: sentences are parsed by assigning them a random binary-branching structure with final punctuation always attached high. The parser could easily be extended to deal with non-right-branching languages by having a number of simple alternative start states (right branching, left branching, random branching, ...). The first step in learning would be to use each start state to parse the training corpus, and choose the start state which results in the best initial-state score.[6]

[4] If a corpus that is bracketed with nonterminal labels is available, then one might ask why not just collect statistics on context-free rules based on node expansions on the annotated corpus. In [102], it is shown that a grammar produced this way is ineffective, in fact performing worse than assigning right linear structure to input sentences.

[5] This is the same output given by systems described in [79, 16, 88, 102].

[6] Note that we do not necessarily have to begin in a *naive* start state. The system could be used as a postprocessor to improve the performance of another parser by using the output of that parser as the

**Structural Transformations**

The next stage involves learning a set of transformations that can be applied to the output of the naive parser to make these sentences better conform to the proper structure specified in the training corpus. The list of possible transformation types is prespecified. Transformations involve making a simple change triggered by a simple environment. In the current implementation, there are twelve allowable transformation types:

- (1-8) $(Add|delete)$ a $(left|right)$ parenthesis to the $(left|right)$ of part of speech tag X.
- (9-12) $(Add|delete)$ a $(left|right)$ parenthesis between tags X and Y.

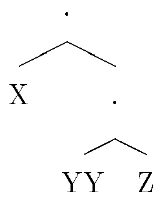
To carry out a transformation by adding or deleting a parenthesis, a number of additional simple structural changes must take place to preserve balanced parentheses and binary branching. To give an example, to delete a left paren in a particular environment, the following operations take place (assuming, of course, that there is a left paren to delete):

1. Delete the left paren.
2. Delete the right paren that matches the just deleted paren.
3. Add a left paren to the left of the constituent immediately to the left of the deleted left paren.
4. Add a right paren to the right of the constituent immediately to the right of the deleted paren.
5. If there is no constituent immediately to the right, or none immediately to the left, then the rule fails to apply.

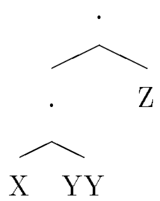
Structurally, the transformation can be seen as follows. If we wish to delete a left paren to the right of constituent X,[7] where X appears in a subtree of the form:

initial state, and then learning corrective transformations. We are using naive start states because this is consistent with our desire to produce a portable system capable of annotating with very little human supervision in the training process.

[7] To the right of the rightmost terminal dominated by X if X is a nonterminal.




carrying out these operations will transform this subtree into:




Given the sentence:[8]

The dog barked .

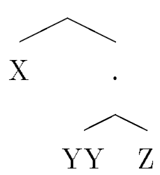
this would initially be bracketed by the naive parser as:

( ( The ( dog barked ) ) . )

If the transformation *delete a left paren to the right of a determiner* is applied, the structure would be transformed to the correct bracketing:

( ( ( The dog ) barked ) . )

To add a right parenthesis to the right of YY, YY must once again be in a subtree of the form:




[8]Input sentences are also labelled with parts of speech.

If it is, the following steps are carried out to add the right paren:

1. Add the right paren.
2. Delete the left paren that now matches the newly added paren.
3. Find the right paren that used to match the just deleted paren and delete it.
4. Add a left paren to match the added right paren.

This results in the same structural change as deleting a left paren to the right of X in this particular structure. While applying different transformations may result in the same structure, each transformation has a different triggering environment. This is significant, because some triggering environments may be better generalizations than others, and so by having a transformation triggered by a number of different environments, the system can find the most effective triggering environment during training.

Applying the transformation *add a right paren to the right of a noun* to the bracketing:

( ( The ( dog barked ) ) . )

will once again result in the correct bracketing:

( ( ( The dog ) barked ) . )

The twelve transformation templates can be broken down into two allowable structural transformations triggered by nine different simple environments. Figure 7.1 shows the two allowable structural operations that can be applied to any subtree, where A, B and C are nonterminals or preterminals.

There are nine different possible triggering environments for transforming a subtree from the left structure in figure 7.1 to the right structure. Say that the subtree to be transformed is as in figure 7.2, where X, A1, A2, B1, B2, C1, C2 and Y are all preterminals. X is the preterminal immediately before the subtree and Y is the preterminal immediately after it. The transformations that result in this structural change are:

1. Add a left paren to the left of A1.

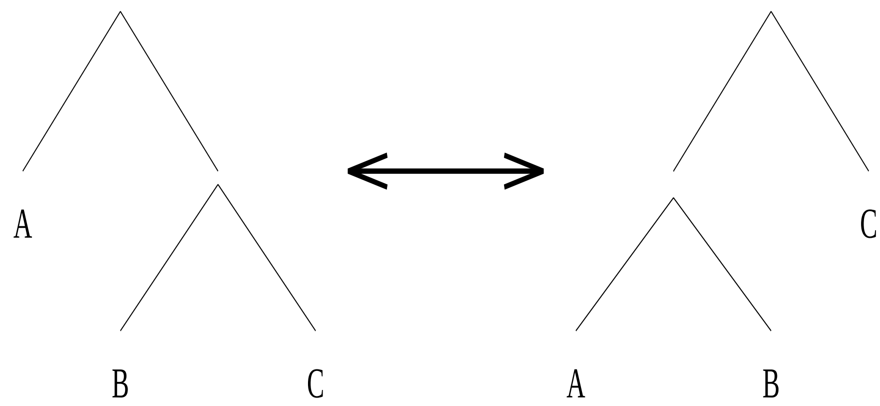



Figure 7.1: Allowable Structural Transformations.

2. Add a right paren to the left of C1.

3. Add a left paren to the right of X.

4. Add a right paren to the right of B2.

5. Delete a left paren to the left of B1.

6. Delete a left paren to the right of A2.

7. Add a left paren between X and A1.

8. Add a right paren between B2 and C1.

9. Delete a left paren between A2 and B1.

In other words, there are nine triggers for transforming this tree. Namely, the value of:

1. A1.

2. C1.

3. X.

4. B2.

5. B1.

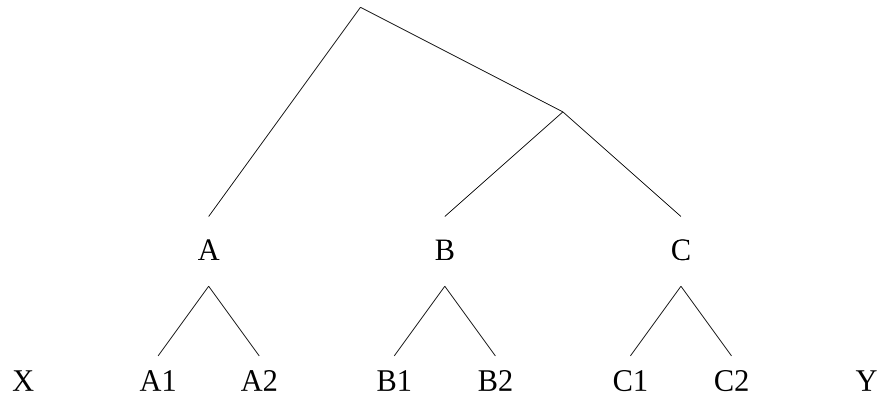



Figure 7.2: Triggering Environments.

6. A2.

7. X and A1.

8. B2 and C1.

9. A2 and B1.

For transforming the tree structure shown in figure 7.3, the nine triggering environments are:

1. C2.

2. A2.

3. Y.

4. B1.

5. B2.

6. C1.

7. Y and C2.

8. A2 and B1.

9. B2 and C1.

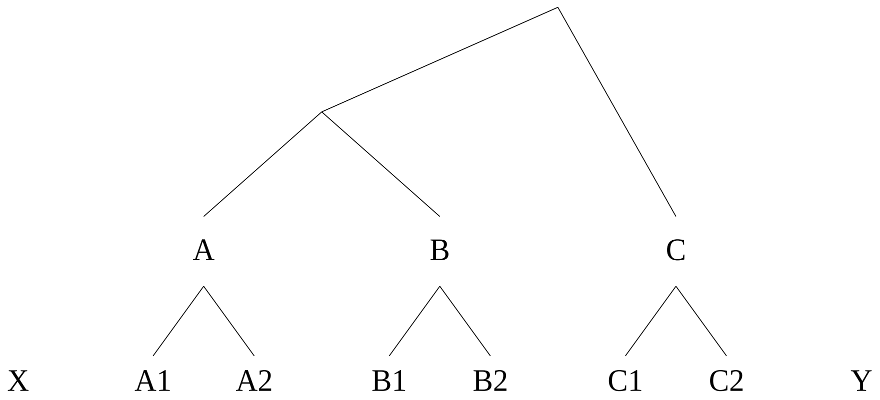



Figure 7.3: Triggering Environments.

**Learning Transformations**

Learning proceeds as follows. Sentences in the training set are first parsed using the naive parser which assigns right linear structure to all sentences, attaching final punctuation high. Next, for each possible instantiation of the twelve transformation templates, that particular transformation is applied to the naively parsed sentences. The resulting structures are then scored using some measure of success which compares these parses to the correct structural descriptions for the sentences provided in the training corpus. The transformation resulting in the best scoring structures (summed over the entire corpus) then becomes the first transformation of the ordered set of transformations that are to be learned. That transformation is then applied to the right-linear structures, and then learning proceeds on the corpus of improved sentence bracketings. The following procedure is carried out repeatedly on the training corpus until no more transformations can be found whose application reduces the error in parsing the training corpus:

1. The best transformation is found for the structures output by the parser in its current state.[9]

2. The transformation is applied to the output resulting from bracketing the corpus using the parser in its current state.

[9]The *state* of the parser is defined as naive initial-state knowledge plus all transformations that currently have been learned.

3. This transformation is added to the end of the ordered list of transformations.

4. Go to 1.

After a set of transformations has been learned, it can be used to effectively parse fresh text. To parse fresh text, the text is first naively parsed and then every transformation is applied, in order, to the naively parsed text. The run time of the learning algorithm is $O(|op| * |env| * |n|)$, where $|op|$ is the number of allowable transformation operations, $|env|$ is the number of possible triggering environments, and $|n|$ is the training corpus size. For each pairing of a transformation operation and environment, the learner must scan the entire corpus and apply the operation whenever a triggering environment is encountered. Of course, as discussed earlier, the actual training run time will be considerably less than the theoretical bound, due to the data-driven nature of the algorithm. As a function of transformation list size $|T|$ and corpus size $|n|$, the run time of the trained annotator is $O(|T| * |n|)$.

One nice feature about this method is that different measures of bracketing success can be used: learning can proceed in such a way as to try to optimize any specified measure of success. When training with the inside-outside algorithm, the stochastic grammar is trained to maximize the probability of the training corpus, in hope that this will result in a grammar that parses fresh text in a way consistent with linguistic intuition. With transformation-based learning, there can be a tighter relationship between the measure that guides learning and the final measure of success. The measure we have chosen for our experiments is the same measure described in [88], which is a variation of a measure which arose out of various meetings on parser evaluation [8]. The measure is the percentage of constituents (strings of words between matching parentheses) from sentences output by our system which do not cross any constituents in the Penn Treebank structural description of the sentence. For example, if our system outputs:

( ( ( The big ) ( dog ate ) ) . )

and the Penn Treebank bracketing for this sentence was:

( ( ( The big dog ) ate ) . )

then the constituent *the big* would be judged correct whereas the constituent *dog ate* would not.

In figure 7.4, we show the first twenty transformations found from one run of training on the Wall Street Journal corpus, which was initially bracketed using the right-linear initial-state parser.[10]

| Number | Add/Delete | Left/Right Paren | Environment |
|---|---|---|---|
| 1 | Delete | Left | Left of NN |
| 2 | Delete | Left | Left of NNS |
| 3 | Add | Right | Left of , |
| 4 | Delete | Left | Between NNP and NNP |
| 5 | Delete | Left | Right of DT |
| 6 | Add | Right | Left of , |
| 7 | Delete | Right | Left of NNS |
| 8 | Delete | Right | Between NN and NN |
| 9 | Delete | Left | Between JJ and JJ |
| 10 | Delete | Left | Right of $ |
| 11 | Add | Right | Between NN and , |
| 12 | Delete | Left | Left of POS |
| 13 | Add | Right | Between NNP and IN |
| 14 | Delete | Left | Between CD and CD |
| 15 | Delete | Left | Between NNP and NNP |
| 16 | Delete | Left | Between JJ and JJ |
| 17 | Add | Right | Left of , |
| 18 | Add | Right | Left of , |
| 19 | Add | Right | Left of , |
| 20 | Delete | Left | Left of " |

Figure 7.4: The first 20 learned transformations.

The first two transformations, as well as transformation number 4, 5, 7, 8, 9, 12, 13, 14, 15 and 16 all extract noun phrases from the right linear initial structure. After bracketing in the initial state, every word will be the leftmost terminal of a phrase containing the entire remainder of the sentence to its right. The first two transformations effectively remove singular and plural common nouns from such a structure and bracket them with the preceding constituent instead. The sentence "The cat meowed ." would initially be

[10] The run was done on 250 training sentences of length 2 to 25. The first 20 out of a total of 160 learned transformations are shown.

bracketed as:

( ( The/DT ( cat/NN meowed/VBD ) ) ./. )

Applying the first transformation to this bracketing (or the second transformation to the same bracketing with *cats* replacing *cat*) would result in:

( ( ( The cat ) meowed ) . )

If there is a left parenthesis between two proper nouns, then the second proper noun is bracketed with constituents that follow it rather than with the preceding proper noun. The fourth transformation fixes this. The sentence *General Motors is very profitable .* would initially be bracketed as:

( ( General/NNP ( Motors/NNP ( is ( very profitable ) ) ) ) . )

Applying the fourth transformation would convert this structure to:

( ( ( General Motors ) ( is ( very profitable ) ) ) . )

The following example demonstrates the interaction and ordering of transformations. The sentence *The fastest car won .* would initially be bracketed as:

( ( The/DT ( fastest/JJ ( cars/NNS won/VBD ) ) ) . )

The first transformation to apply to this sentence would be number 2, resulting in:

( ( The ( ( fastest cars ) won ) ) . )

The next applicable transformation is number 5, whose application results in:

( ( ( The ( fastest cars ) ) won ) . )

After this transformation is applied, no other transformations can be applied to the sentence, and the correct structure is produced.

Transformation number 10 results from the fact that a number usually follows a dollar sign, and these two lexical items should be bracketed together. Transformations 3, 6, 11, 17, 18 and 19 result from the fact that a comma is a good indicator of the preceding phrase being terminated. Since each transformation is carried out only once per environment, multiple listings of a transformation are required if the transformation is to be applied multiple times. The sentence *We called them , but they were gone .* would initially be bracketed as:

( ( We/PP ( called/VBD ( them/PP ( ,/, ( but ( they left ) ) ) ) ) ) . )

The first applicable transformation is number 3, whose application results in:

( ( We ( ( called them ) ( , ( but ( they left ) ) ) ) ) . )

The next applicable transformation is number 6, whose application results in the correct structure:

( ( ( We ( called them ) ) ( , ( but ( they left ) ) ) ) . )

### 7.1.2 Results Using Manually Tagged Text

In the first experiment we ran, training and testing were done on the Texas Instruments Air Travel Information System (ATIS) corpus [56].[11] In table 7.1, we compare results obtained using transformation-based error-driven learning to results cited in [88] using the inside-outside algorithm on the same corpus. Accuracy is measured in terms of the percentage of noncrossing constituents in the test corpus, as described above. Our system was tested by using the training set to learn a set of transformations, and then applying these transformations to the test set and scoring the resulting output. We then used the jacknife approach (see [113]) to estimate the variance of our result from a single run of learning and applying transformations. Doing so, we compute that the 95% confidence interval for this experiment is $91.1\% \pm 2.1\%$. In this experiment, 64 transformations were learned (compared with 4095 context-free rules and probabilities used in the inside-outside algorithm experiment). It is significant that we obtained comparable performance using a training corpus only 21% as large as that used to train the inside-outside algorithm.

[11] In all experiments described here and elsewhere, results are calculated on a test corpus which was not used in any way in either training the learning algorithm or in developing the system.

| Method | # of Training Corpus Sentences | Accuracy |
|---|---|---|
| Inside-Outside | 700 | 90.4% |
| Transformation-Learner | 150 | 91.1% |

Table 7.1: Comparing two learning methods on the ATIS corpus.

After applying all learned transformations to the test corpus, 60% of the sentences had no crossing constituents, 74% had fewer than two crossing constituents, and 85% had fewer than three. In [12], training and testing were also done on the ATIS corpus, with 96% of sentences in the test set were parsed exactly correctly (a much more difficult task) when training on 675 sentences. However, when training on only 150 sentences, accuracy is approximately 30%. In addition, it is unclear whether this technique can be effectively used on corpora that are structurally more varied and complex.

The mean sentence length of the test corpus was 11.3. In figure 7.5, we have graphed percentage correct as a function of the number of transformations that have been applied to the test corpus. As the transformation number increases, overtraining sometimes occurs. In the current implementation of the learner, a transformation is added to the list if it results in *any* positive net change in the training set. Toward the end of the learning procedure, transformations are found that only affect a very small percentage of training sentences. Since small counts are less reliable than large counts, we cannot reliably assume that these transformations will also improve performance in the test corpus. One way around this overtraining would be to set a threshold: specify a minimum level of improvement that must result for a transformation to be learned. Another possibility is to use additional training material to prune the set of learned transformations.

We next ran an experiment to determine what performance could be achieved if we dropped the initial right-linear assumption. Using the same training and test sets as above, sentences were initially assigned a random binary-branching structure, with final punctuation always attached high. Since there was less regular structure in this case than in the right-linear case, many more transformations were found, 147 transformations in total. When these transformations were applied to the test set, a bracketing accuracy of

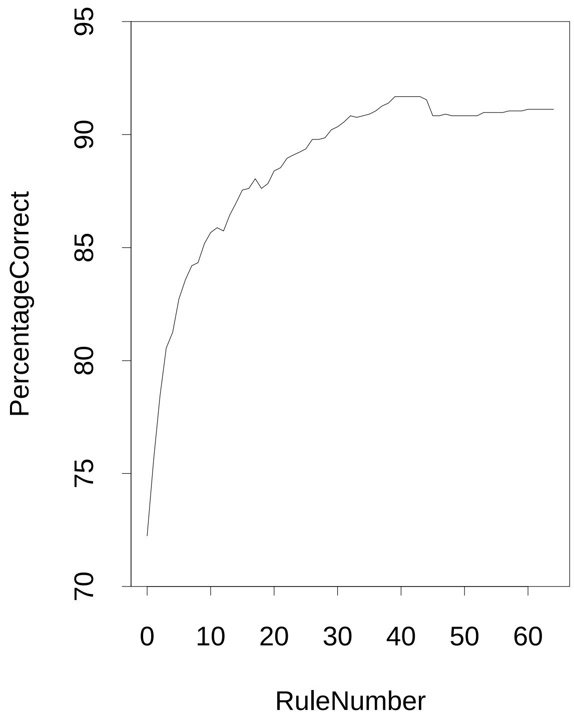



Figure 7.5: Results From the ATIS Corpus, Starting With Right-Linear Structure

87.1% resulted. The graph of this is shown in figure 7.6.

The ATIS corpus is structurally fairly regular. To determine how well our algorithm performs on a more complex corpus, we next ran experiments on the Wall Street Journal. Results from this experiment can be found in table 7.2. [12] Accuracy is again measured as the percentage of constituents in the test set which do not cross any Penn Treebank constituents.[13]

In the corpus we used for the experiments of sentence length 2-15, the mean sentence length was 10.8. In the corpus used for the experiment of sentence length 2-25, the mean length was 16.8. As would be expected, performance degrades somewhat as sentence length increases. In table 7.3, we show the percentage of sentences in the test corpus which have

[12] For sentences of length 2-15, the initial right-linear parser achieves 69% accuracy. For sentences of length 2-20, 63% accuracy is achieved and for sentences of length 2-25, accuracy is 59%.

[13] In all of our experiments carried out on the Wall Street Journal, the test set was a randomly selected set of 500 sentences.

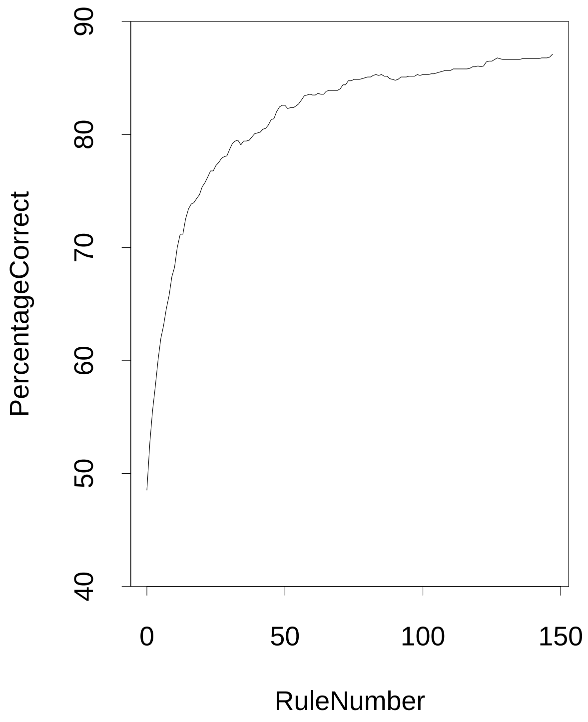



Figure 7.6: Results From the ATIS Corpus, Starting With Random Structure

no crossing constituents, and the percentage that have only a very small number of crossing constituents.[14]

In table 7.4, we show the standard deviation measured from three different randomly chosen training sets of each sample size and randomly chosen test sets of 500 sentences each, as well as the accuracy as a function of training corpus size for sentences of length 2 to 20.

A graph showing parsing performance for the WSJ run trained on a 500-sentence training corpus (training and testing on sentences of length 2-15) is shown in figure 7.7.

In [102], an experiment was done using the inside-outside algorithm to train a context-free grammar from the partially bracketed Wall Street Journal corpus. As in the experiment

[14] For sentences of length 2-15, the initial right linear parser parses 17% of sentences with no crossing errors, 35% with one or fewer errors and 50% with two or fewer. For sentences of length 2-25, 7% of sentences are parsed with no crossing errors, 16% with one or fewer, and 24% with two or fewer.

| Sent. Length | # Training Corpus Sents | # of Transformations | % Accuracy |
|---|---|---|---|
| 2-15 | 250 | 83 | 88.1 |
| 2-15 | 500 | 163 | 89.3 |
| 2-15 | 1000 | 221 | 91.6 |
| 2-20 | 250 | 145 | 86.2 |
| 2-25 | 250 | 160 | 83.8 |

Table 7.2: WSJ Sentences

| Sent. Length | # Training Corpus Sents | % of 0-error sents | % of ≤1-error sents | % of ≤2-error sents |
|---|---|---|---|---|
| 2-15 | 1000 | 62.4 | 77.2 | 87.8 |
| 2-25 | 250 | 29.2 | 44.9 | 59.9 |

Table 7.3: WSJ Sentences.

with the ATIS corpus, all possible binary context-free rules were initially allowed, and random probabilities were initially assigned to each rule. A comparison of this approach to the transforamtion-based approach is shown in tables 7.5 and 7.6. The inside-outside experiment was carried out on sentences of length 1-15, and the transformation-based approach was carried out on sentences of length 2-15. The inside-outside experiment had a grammar of 4095 probabilistic context free rules, which could be trimmed down to 450 rules without changing performance. 221 symbolic transformations were learned in the transformation-based experiment. In table 7.5, the transformation-based learner

| # Training Corpus Sents | % Correct | Std. Dev. |
|---|---|---|
| 0 | 63.0 | 0.69 |
| 10 | 75.8 | 2.95 |
| 50 | 82.1 | 1.94 |
| 100 | 84.7 | 0.56 |
| 250 | 86.2 | 0.46 |
| 750 | 87.3 | 0.61 |

Table 7.4: WSJ Sentences of Length 2 to 20.

is shown to outperform the inside-outside algorithm when parsing accuracy is measured in terms of crossing brackets. Applying the jacknife method to estimate the variance of our result from the single training and testing run, the 95% confidence interval obtained was $91.6\% \pm 1.2\%$. In table 7.6, accuracy is measured as the percentage of sentences with no crossing bracket violations. Once again applying the jacknife method to estimate the variance of our sentence accuracy gives a 95% confidence interval of $62.4\% \pm 4.3\%$.

We believe it is significant that comparable performance was obtained, considering that the transformation-based approach is only a *weakly* statistical learner (only integer addition and comparison is done in learning) and is a completely symbolic parser that can parse in linear time.

| Method | # of Training Sents. | # of Transforms | % Accuracy |
|---|---|---|---|
| Inside-Outside | 1095 | 4095/450 | 90.2 |
| Transformation-Based | 1000 | 221 | 91.6 |

Table 7.5: Comparing Two Approaches - Crossing Bracket Measure

| Method | Sentence Accuracy |
|---|---|
| Inside-Outside | 57.1 |
| Transformation-Based | 62.4 |

Table 7.6: Comparing Two Approaches - Sentence Accuracy

We also ran an experiment on WSJ sentences of length 2-15 starting with random binary-branching structures with final punctuation attached high. In this experiment, 325 transformations were found, and the accuracy resulting from applying these transformations to a test set was 84.7%. In figure 7.8 we show the sentence length distribution in the Wall Street Journal corpus.

In table 7.7 and table 7.8, we show results from running the bracketing algorithm on the Penn Treebank bracketing of the Brown Corpus. In each run, a different randomly chosen training and test set was used. Each test set contained 500 sentences. The slightly lower crossing bracket accuracy on this corpus compared to the Wall Street Journal is

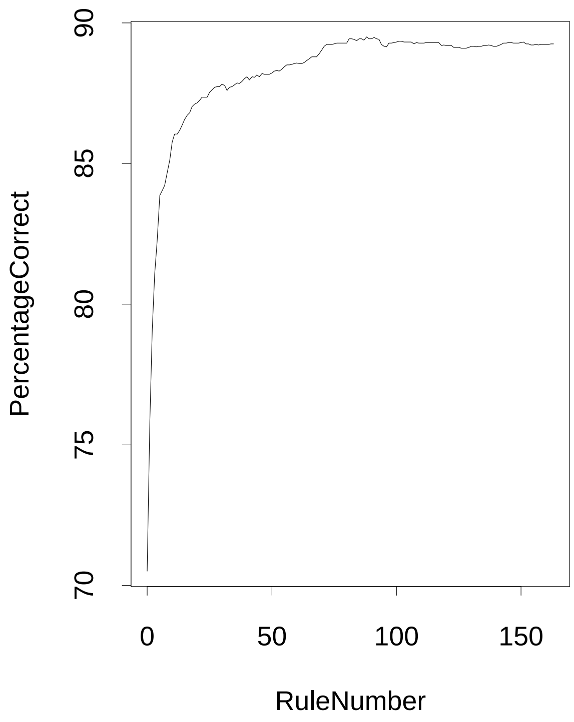



Figure 7.7: Results From the WSJ Corpus

likely due to the more varied nature of the corpus.

We also ran one experiment using the Old English corpus. Both the training and test set were 210 sentences. All sentences were between 2 and 25 words long. In total, 101 transformations were learned. Initially assigning right-branching structure to the test set resulted in an accuracy of 56.8%. After applying the transformations, bracketing accuracy improved to 88.7%. The first fifteen learned transformations are shown in figure 7.9.

### 7.1.3 Results Using Automatically Tagged Text

While the numbers presented above allow us to compare the transformation learner with systems trained and tested on comparable corpora, these results are all based upon the assumption that the test data is tagged fairly reliably (manually tagged text was used in all of these experiments, as well in the experiments of [88, 102].) When parsing free text,

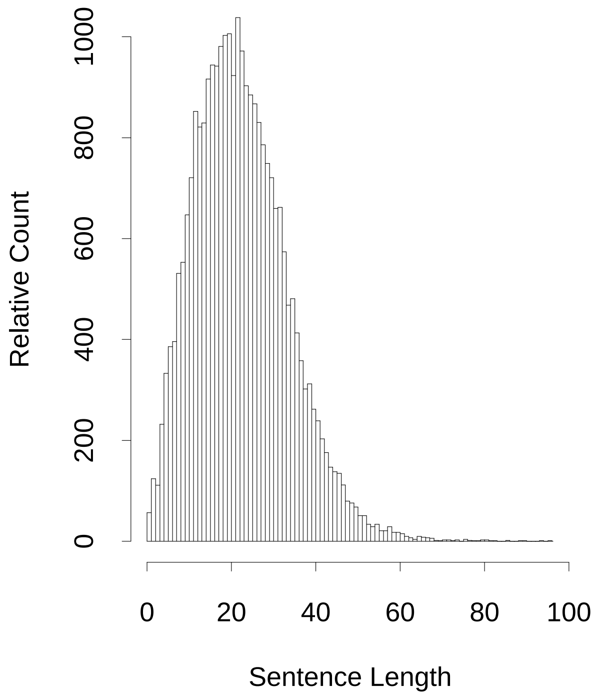



Figure 7.8: The Distribution of Sentence Lengths in the WSJ Corpus.

we cannot assume that the text will be tagged with the accuracy of a human annotator. Instead, an automatic tagger would have to be used to first tag the text before parsing. To address this issue, we first ran one experiment where we randomly induced a 5% tagging error rate beyond the error rate of the human annotator. Errors were induced in such a way as to preserve the unigram part of speech tag probability distribution in the corpus. The experiment was run for sentences of length 2-15, with a training set of 1000 sentences and a test set of 500 sentences. The resulting bracketing accuracy was 90.1%, compared to 91.6% accuracy when using an unadulterated training corpus. Accuracy only degraded by a small amount when training on the corpus with adulterated part of speech tags, suggesting that high parsing accuracy rates could be achieved if tagging of the input were done automatically by a part of speech tagger.

Next, we ran experiments on two different randomly selected training and testing on

| # Training Corpus Sents | Run Number | % Correct |
|---|---|---|
| 250 | 1 | 83.5 |
| 250 | 2 | 85.5 |
| 750 | 1 | 85.1 |
| 750 | 2 | 85.0 |

Table 7.7: Brown Corpus Sentences of Length 2 to 20.

| # Training Corpus Sents | Run Number | % of 0-error sents | % of $\leq$1-error sents | % of $\leq$2-error sents |
|---|---|---|---|---|
| 250 | 1 | 37.4 | 59.6 | 71.0 |
| 250 | 2 | 42.4 | 61.0 | 71.6 |
| 750 | 1 | 41.8 | 58.0 | 70.4 |
| 750 | 2 | 40.6 | 58.0 | 71.6 |

Table 7.8: Brown Corpus Sentences of Length 2 to 20.

500 sentences of length 2 to 20 from the Wall Street Journal. First, the Penn Treebank part of speech tags were used in both the training corpus and the test corpus. Next, the training corpus tags were taken from the Penn Treebank, but the test corpus tags were obtained by automatically tagging the corpus using the tagger described in the previous chapter. Last, both the training and test set were automatically tagged. Results from applying the tagging transformations to the two training and two test corpora are shown in table 7.9. Because the tagging accuracy on these corpora is worse than was achieved in the previous section, we also ran the statistical tagger on these corpora for a comparison of tagging accuracy. Apparently, the shorter sentences are more difficult for both taggers (these corpora are restricted to sentences of length 2-20, whereas the previous corpora were not). In all cases, the transformation-based tagger outperformed the stochastic tagger. The bracketing results are shown in table 7.10. We are encouraged that performance degradation is not significant when automatically tagging as compared to using human-annotated text.

Below are ten randomly chosen sentences from the above experiment. In each case, the

| Number | Add/Delete | Left/Right Paren | Environment |
|---|---|---|---|
| 1 | Delete | Left | Left of NN |
| 2 | Add | Right | Right of NN |
| 3 | Add | Right | Left of , |
| 4 | Add | Left | Left of CO |
| 5 | Add | Left | Right of VT |
| 6 | Add | Right | Right of VT |
| 7 | Add | Right | Left of , |
| 8 | Add | Left | Right of VT |
| 9 | Add | Left | Right of , |
| 10 | Add | Right | Left of ; |
| 11 | Add | Right | Left of ; |
| 12 | Add | Left | Left of PR |
| 13 | Delete | Left | Between RB and RB |
| 14 | Add | Right | Left of ; |
| 15 | Delete | Left | Right of PR |

Figure 7.9: The first 15 learned transformations for bracketing Old English.

output of the bracketing program is listed first, and the Penn Treebank bracketing is listed second. Crossing brackets are marked with a star.

( ( But ( if *( ( a raider ) *( takes *( ( over ( when ( ( the stock ) ( is weak ) ) ) ) ( , ( ( the shareholder ) ( never ( gets ( his recovery ) ) ) ) ) )* )* )* ) ) . )
( ( But ( if ( ( a raider ) ( takes over ( when ( ( the stock ) ( is weak ) ) ) ) ) ) , ( ( the shareholder ) ( never gets ( his recovery ) ) ) ) . )

( ( ( The company ) ( expects ( to ( resume *( ( full operations ) ( by today ) )* ) ) ) ) . )
( ( ( The company ) ( expects ( to ( resume ( full operations ) ) ( by today ) ) ) ) . )

( ( ( “ It ) *( ( ’s *( ( very likely ) *( ( ( ( the next ) ( five years ) ) ( will ( be ( strong ( for funds ) ) ) ) ) ) ( , ” ) )* )* ) ( he says ) )* ) . )
( ( ( “ It ( ’s ( very likely ( ( the next five years ) will ( be strong ( for funds ) ) ) ) ) , ” ) ( he says ) ) . )

( ( ( The ( latest report ) ) ( compares ( with ( ( ( a modest ) ( 9.9 ( % increase ) ) ) ( in *( ( July ( machine orders ) ) ( from ( ( a year ) earlier ) ) )* ) ) ) ) ) . )
( ( ( The latest report ) ( compares ( with ( a modest 9.9 % increase ( in ( July machine

| Method | Corpus | Unknown Word Accuracy | Known Word Accuracy | Total Accuracy |
|---|---|---|---|---|
| Transformations | Test 1 | 75.3 | 94.7 | 90.8 |
| Statistical | Test 1 | 65.1 | 94.7 | 88.9 |
| Transformations | Train 1 | 79.2 | 95.1 | 92.1 |
| Statistical | Train 1 | 65.3 | 95.2 | 89.4 |
| Transformations | Test 2 | 78.5 | 94.6 | 91.4 |
| Statistical | Test 2 | 64.0 | 94.8 | 88.7 |
| Transformations | Train 2 | 79.2 | 95.1 | 92.0 |
| Statistical | Train 2 | 65.0 | 94.7 | 88.9 |

Table 7.9: Accuracy In Tagging The Training and Test Corpora.

| Experiment | Tags Used In Training | Tags Used In Testing | % Correct |
|---|---|---|---|
| 1 | Human | Human | 87.1 |
| 1 | Human | Automatic | 85.7 |
| 1 | Automatic | Automatic | 85.8 |
| 2 | Human | Human | 88.1 |
| 2 | Human | Automatic | 86.6 |
| 2 | Automatic | Automatic | 86.5 |

Table 7.10: Crossing Bracket Accuracy on WSJ Sentences of Length 2 to 20.

orders ) ) ( from ( ( a year ) earlier ) ) ) ) ) ) . ) The latest report compares with a modest 9.9 orders from a year earlier .

( ( ( The goal ) ( was ( to ( boost ( ( the circulation ) ( above ( ( the ( 500,000 level ) ) ( ( considered significant ) ( by advertisers ) ) ) ) ) ) ) ) ) ) . )
( ( ( The goal ) ( was ( to ( boost ( the circulation ) ( above ( ( the 500,000 level ) ( considered significant ( by advertisers ) ) ) ) ) ) ) ) . )

( ( ( Mr. Jones ) ( ran *( ( ( ( for ( the Senate ) ) ( as ( a Democrat ) ) ) ( in 1986 ) ) ( , ( but ( lost ( to ( ( incumbent Sen. ) ( Don Nickles ) ) ) ) ) ) )* ) ) . )
( ( ( Mr. Jones ) ( ( ran ( for ( the Senate ) ) ( as ( a Democrat ) ) ( in 1986 ) ) , but ( lost ( to ( incumbent Sen. Don Nickles ) ) ) ) ) . )

( ( Then ( ( ( the ( ( auto paint ) shop ) ) fire ) ( sent ( ( an ( evil-looking cloud ) ) *( of

*( ( black smoke ) ( into ( the air ) ) )* )* ) ) ) ) . )
( ( Then ( ( the auto paint shop fire ) ( sent ( an evil-looking cloud ( of ( black smoke ) ) ) ( into ( the air ) ) ) ) ) . )

( ( He ( *( used *( to ( be ( ( a boiler-room ) salesman ) ) )* )* ( , ( peddling ( investments ( *( in oil )* *( *( ( and ( gas wells ) ) and )* ( rare coins ) )* ) ) ) ) ) ) . )
( ( He ( used ( to ( be ( a boiler-room salesman ) , ( peddling ( investments ( in ( ( oil and gas wells ) and ( rare coins ) ) ) ) ) ) ) ) ) . )

( ( ( The board ) ( is ( scheduled ( to ( meet Tuesday ) ) ) ) ) ) . )
( ( ( The board ) is ( scheduled ( to ( meet Tuesday ) ) ) ) . )

( ( Ignore ( the ( present condition ) ) ) . )
( ( Ignore ( the present condition ) ) . )

In the first example, there are three bracketing errors, all arising from the failure to end the clause following *if* at the comma. The second sentence has one error, which is a prepositional phrase attachment error. The third sentence has three bracketing errors, arising from crossing matching quotes. Perhaps a number of meta-rules, either learned or manually coded, such as information about matching parentheses and quotes, would significantly improve performance. The fourth sentence has one error, which is again a prepositional phrase attachment error. The sixth sentence has one error, from attaching the clause following (and including) the comma to the preposition *for* instead of the verb *ran*. The seventh sentence has two errors, both due to prepositional phrase attachment. The eighth sentence has five errors, one of which is due to prepositional phrase attachment and two arising from a difficult coordinate structure. In addition to meta-rules, postprocessors addressing particular parsing problems such as prepositional phrase attachment and coordination could lead to significant system performance improvements. In a later section, we will discuss a transformation-based prepositional phrase attachment postprocessor.

### Conclusions

In this section, we have described a new approach for learning a grammar to automatically parse text. The method can be used to obtain high parsing accuracy with a very small training set. Instead of learning a traditional grammar, an ordered set of structural transformations is learned that can be applied to the output of a very naive parser to obtain binary-branching trees with unlabelled nonterminals. Experiments have shown that these parses conform with high accuracy to the structural descriptions specified in a manually annotated corpus. Unlike other recent attempts at automatic grammar induction that rely heavily on statistics both in training and in the resulting grammar, our learner is only very weakly statistical. For training, only integers are needed and the only mathematical operations carried out are integer addition and integer comparison. The resulting grammar is completely symbolic. Unlike learners based on the inside-outside algorithm which attempt to find a grammar to maximize the probability of the training corpus in hope that this grammar will match the grammar that provides the most accurate structural descriptions, the transformation-based learner can readily use any desired success measure in learning.

In the future, we plan to experiment with other types of transformations. Currently, each transformation in the learned list is only applied once in each appropriate environment. For a transformation to be applied more than once in one environment, it must appear in the transformation list more than once. One possible extension to the set of transformation types would be to allow for transformations of the form: add/delete a paren as many times as is possible in a particular environment. In addition, each transformation is applied to triggering environments in a sentence strictly right to left or left to right. Doubling the list of transformations to allow a transformation to apply in either direction might prove effective. To expand the system to allow for non-binary branching structure, a transformation that adds and deletes structure could be added, although doing so would require a more sophisticated measure to guide the learning process. We also plan to experiment with other scoring functions and control strategies for finding transformations and to use this system as a postprocessor to other grammar induction systems, learning transformations to improve their performance. In addition, we plan to incorporate more lexical information into the learner. Currently, the only lexical information used is the part of speech tag of each word. Lexical information could be incorporated by allowing transformations to reference the n most frequently occurring words. Or, words could be labelled with features and transformations could make reference to features in addition to part of speech tags.[15] We hope these future paths will lead to a trainable and very accurate parser for free text.

In the following section we describe a method for labelling the nonterminal nodes of a syntactic tree. The parser will first use the *transformational grammar* to output a parse tree without nonterminal labels, and then a separate algorithm will be applied to that tree to label the nonterminals.

## 7.2 Labelling Nonterminal Nodes

Once a tree is bracketed, the next step is to label the nonterminal nodes. Transformation-based error-driven learning is once again used for learning how to label nonterminals. Currently, a node is labelled based solely on the labels of its daughters. Therefore, an unlabelled tree can be labelled in a bottom-up fashion. Instead of addressing the problem of labelling the unlabelled tree output of the previous section, we have addressed a slightly different problem. The problem is to assign a tag to a node of a properly bracketed tree given the proper labels for the daughter nodes. This problem can be more easily evaluated and solving it is a significant step toward solving the problem of labelling the output of the transformation-based bracketer.

This experiment used the Penn Treebank bracketed Wall Street Journal corpus.[16] Two training sets were used (training set A had 1878 sentences and training set B had 1998), as well as a test set of 1971 sentences. A total of 19 nonterminal symbols occurred in the training and test sets. In the first experiment, the initial state annotator assigned the label *noun phrase* to all nodes. Then, transformations were learned to improve accuracy. The transformation templates are:

1. Change the node label to X if Y is a daughter.[17]

2. Change the node label to X if Y and Z are adjacent daughters.

Transformations were learned using training set A. A total of 115 transformations were learned. Initially assigning the label *noun phrase* to all nonterminal nodes in the

[15] Incorporating a feature-based lexicon into the learner is discussed in a later section.
[16] Thanks to Rich Pito for providing me with corpus processing tools for running this experiment.
[17] Y can be a nonterminal or preterminal (and need not be the only daughter).

| Transformation Number | Tag As | If Daughter Includes |
|---|---|---|
| 1 | PP | IN |
| 2 | S | VP |
| 3 | VP | VBD |
| 4 | VP | VB |
| 5 | VP | VBN |
| 6 | VP | VBG |
| 7 | VP | VBZ |
| 8 | S | , S |
| 9 | S | VP |
| 10 | SBar | -NONE- S |
| 11 | PP | TO NP |
| 12 | SBar | IN S |
| 13 | VP | VBP |
| 14 | S | VP |
| 15 | S | CC S |
| 16 | WHNP | WDT |
| 17 | SBar | WHNP |
| 18 | VP | CC VP |
| 19 | WHNP | WP |
| 20 | ADJP | JJR |

Table 7.11: Transformations For Labelling Nonterminals.

test set resulted in an accuracy of 44.9%. Applying all learned transformations to the test set resulted in an accuracy of 94.3%. Figure 7.11 shows the first twenty learned transformations.[18] Transformations 15 and 18, as well as a number of similar transformations in the entire list capture the general rule $X \rightarrow$ X and X for coordination. It appears that the transformation *Change a label to S if VP is a daughter* is particularly effective, appearing as transformation 2, 9 and 14. After the second transformation is applied, the transformations that follow could undo the second transformation as a side-effect. So, this transformation applies a number of times to remedy this.

Next, a less naive start state was used. A nonterminal node is assigned the most likely tag for its daughters, as indicated in a second training set (training set B). Unseen daughter sequences are tagged with a default tag (noun phrase). Transformations were learned after applying the start state annotator to training set A. On the test set, initial state accuracy

[18] A listing of Penn Treebank nonterminal labels can be found in appendix D.

| Labelled As | Should Be | Daughters | % of Total Error | Cumulative Error |
|---|---|---|---|---|
| S | NP | NP VP | 13.7 | 13.7 |
| Sbar | Sbarq | WHNP S | 4.7 | 18.4 |
| Sbar | PP | IN S | 3.1 | 21.5 |
| S | ADVP | NP | 2.1 | 23.6 |
| VP | ADJP | VBN | 1.4 | 25.0 |
| X | ADVP | RB NP | 1.3 | 26.3 |
| ADVP | ADJP | JJ NP | 1.3 | 27.6 |
| Sbar | S | PP S | 1.2 | 28.8 |
| NP | PP | -NONE- | 1.2 | 30.0 |
| PP | ADVP | IN PP | 1.1 | 31.1 |

Table 7.12: Top 10 Labelling Errors.

was 92.6%. Applying the transformations resulted in an accuracy of 95.9%. A total of 107 transformations were learned. Figure 7.12 shows the ten costliest errors made on the test set after all transformations are applied. The most common error results from mislabelling certain (NP VP) structures as S rather than NP. One example is shown below, where the system incorrectly labels *the loan guarantees approved yesterday* as an S.

( NP the size ( PP of ( NP ( NP the loan guarantees) ( VP approved yesterday ) ) ) )

We are very encouraged by the accuracy obtained using such a simple learning algorithm that only makes use of very local environments without recourse to any lexical information. Hopefully, adding richer environments such as *the word X is a daughter*, or *the nonterminal to the left is Y* will lead to an even more accurate nonterminal labeller. By first bracketing text and then labelling nonterminals, we can produce labelled parse trees in linear time with respect to sentence length. The bracketer runs in $O(|n| * |T|)$, where $|n|$ is the length of the sentence and $|T|$ is the number of bracketing transformations. The nonterminal labeller also runs in $O(|n| * |T|)$, as all transformations are tried at every nonterminal node. Therefore, parsing run time is: $O(|n| * |T|) + O(|n| * |T|) = O(|n| * |T|)$.

## 7.3 Transformation-Based Postprocessing

Once a sentence has been annotated, it can be further processed by other modules that attempt to correct particular structure types or use information not used in previously applied learning modules to further improve annotation accuracy. As one example of such a postprocessor, we have done some work on a transformation-based prepositional phrase attachment module.[19] Since the previous bracketing module makes use of no lexical information beyond part of speech tags, it is unlikely that it will be able to resolve prepositional phrase attachment with high accuracy. From the sample output of the bracketer shown above, it can be seen that prepositional phrase attachment is one of the most common errors in bracketing. A postprocessor could be used whose set of allowable transformations allows for the movement of prepositional phrases to different attachment locations.

The prepositional phrase attachment module (also discussed in [25]) learns transformations from a corpus of 4-tuples of the form: (v n1 p n2), where v is the matrix verb, n1 is the head of the object noun phrase, p is the preposition and n2 is the head of the noun phrase governed by the preposition (for example, *see/v the boy/n1 on/p the hill/n2*). For all sentences that conform to this pattern in the Wall Street Journal corpus, a 4-tuple was formed.[20] There were 12,766 4-tuples in all, which were randomly split into 12,266 training samples and 500 test samples. In this experiment, the attachment choice for prepositional phrases was between the object noun and the matrix verb. In the initial state annotator, all prepositional phrases are attached to the object noun.[21] This is the attachment predicted by right association [68]. The allowable transformations are:

- Change the attachment location from X to Y if:
  - n1 is Z
  - n2 is Z
  - v is Z
  - p is Z

[19] This work was done with Philip Resnik.

[20] These were extracted by Philip Resnik using tgrep, a tool written by Rich Pito. The 4-tuples were extracted automatically, and mistakes were not manually pruned out.

[21] If it is the case that attaching to the verb would be a better start state in some corpora, this decision could be parameterized.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | N | V | P is *at* |
| 2 | N | V | P is *as* |
| 3 | N | V | P is *into* |
| 4 | N | V | P is *from* |
| 5 | N | V | P is *with* |
| 6 | N | V | N2 is *year* |
| 7 | N | V | P is *by* |
| 8 | N | V | P is *in* and N1 is *amount* |
| 9 | N | V | P is *through* |
| 10 | N | V | P is *during* |
| 11 | N | V | V is *put* |
| 12 | N | V | N2 is *month* |
| 13 | N | V | P is *under* |
| 14 | N | V | P is *after* |
| 15 | N | V | V is *have* and P is *in* |
| 16 | N | V | P is *without* |
| 17 | V | N | P is *of* |
| 18 | N | V | V is *buy* and P is *for* |
| 19 | N | V | P is *before* |
| 20 | N | V | V is *have* and P is *on* |

Figure 7.10: The first 20 transformations learned for prepositional phrase attachment.

– etc. for all members of the power set of (n1,n2,v,p) with $\leq 3$ members.

A total of 471 transformations were learned. In figure 7.10 we show the first 20 transformations that were learned. Initial accuracy is 64.0% when prepositional phrases are always attached to the object noun in the test set. After applying the transformations, accuracy increases to 80.8%.

In the above experiment, all transformations are triggered by words or groups of words. It is surprising that in spite of the inevitable sparse data problems, very good performance is achieved. There are a couple of ways to address the sparse data problem. In this case, mapping words to part of speech will not help. Instead, semantic class information is necessary. One method is to use a manually constructed semantic hierarchy such as that described in [85]. Every word can then be expanded to a list of classes it occurs in, and transformations can be triggered by words and word classes. Another approach is to build a word similarity tree as described in the previous chapter, assign each node in the similarity tree a unique name, and then allow transformations to be triggered by words and node names. Each node name is a feature, and for each feature x, a word is $+x$ if it is a descendent of the node labelled x, and is $-x$ otherwise.

We incorporated the idea of using semantic information in the following way. Using the Wordnet noun hierarchy [85], each noun in the training and test set was replaced by a set containing the noun and the name of every class that noun appears in. The transformation set is modified so that instead of asking if a noun is $X$, it can ask if $X$ is a member of the noun's class set.[22] In [94], a method is proposed for using Wordnet in conjunction with a corpus to obtain class-based statistics. Our method here is much simpler, in that we are only using boolean values to indicate the classes a word can be a member of. Since the transformation-based approach with classes can generalize in a way that the approach without classes is unable to, we would expect fewer transformations to be necessary. This is indeed the case. Training and testing were carried out on the same samples as in the previous experiment. A total of 266 transformations were learned. Applying these transformations to the test set resulted in an accuracy of 81.8%. In figure 7.11 we show the first 20 transformations learned using noun classes. Class descriptions are surrounded by square brackets. The first transformation states that if N2 is a noun that describes time (is a member of the *time* class), then it should be attached to the verb, since time is much more likely to modify a verb (leave the meeting in an hour) than a noun. This experiment also demonstrates how any feature based lexicon can trivially be incorporated into the learner, by extending transformations to allow them to make reference to a word and any of its features.

In [59], Hindle and Rooth use t-score statistics to measure the strength of lexical associations between the preposition and the noun and between the preposition and the verb, and attach prepositional phrases according to these scores. The t-scores are estimated from a set of sentences with a prepositional phrase either attached to the verb or the object noun. This is a superset of the training instances used by the transformation-based method, as it includes sentences (noun phrase fragments) without verbs. Since the corpus was not

[22] For reasons of run-time efficiency, transformations making reference to the classes of both N1 and N2 were not permitted.

| | Change Tag | | |
|---|---|---|---|
| # | From | To | Condition |
| 1 | N | V | N2 is *[time]* |
| 2 | N | V | P is *at* |
| 3 | N | V | P is *as* |
| 4 | N | V | P is *into* |
| 5 | N | V | P is *from* |
| 6 | N | V | P is *with* |
| 7 | N | V | P is *of* |
| 8 | N | V | P is *in* and N1 is *[measure,quantity,amount]* |
| 9 | N | V | P is *by* and N2 is *[abstraction]* |
| 10 | N | V | P is *through* |
| 11 | N | V | P is *in* and N1 is *[group,grouping]* |
| 12 | V | N | V is *be* |
| 13 | N | V | V is *put* |
| 14 | N | V | P is *under* |
| 15 | N | V | P is *in* and N1 is *[written communication]* |
| 16 | N | V | P is *without* |
| 17 | N | V | P is *during* |
| 18 | N | V | P is *on* and N1 is *[thing]* |
| 19 | N | V | P is *after* |
| 20 | N | V | V is *buy* and P is *for* |

Figure 7.11: The first 20 transformations learned for prepositional phrase attachment, using noun classes.

bracketed, a heuristic was used to guess the proper prepositional phrase attachment. In their paper, they train on over 200,000 sentences with prepositions from the AP newswire, and they quote an accuracy of 78% . Their algorithm was reimplemented and tested using the same training and test set used for the above experiments.[23] Doing so resulted in an attachment accuracy of 70.4%. Next, the training set was expanded to include the entire Wall Street Journal corpus (including unambiguous attachments but excluding the test set). Accuracy improved to 75.8% using the larger training set, still significantly lower than accuracy obtained using the transformation-based approach.[24] Using the technique described in [94] to attach prepositional phrases based on semantic similarity estimated using Wordnet, an accuracy of 72% was obtained using the same training and test sets. Using the semantic approach in conjunction with the method described by Hindle and Rooth [94] (backing off from the Hindle/Rooth method to the semantic based method) resulted in an accuracy of 76.0%, still lower than the results obtained using transformations. Since the t-score approach did not make reference to n2, we reran the transformation-learner disallowing all transformations that make reference to n2. Doing so resulted in an accuracy of 79.2%. See figure 7.12.

We next used the jacknife technique to estimate the variance of the result obtained using the transformation-learner with word classes. This resulted in a 95% confidence interval accuracy of 81.8% $\pm$ 3.5%.

As was previously mentioned, extending the learner is a trivial task. In the near future, we intend to incorporate information about the subject head, as well as Wordnet class information about the verb, into the learner.

[23] Attachment heuristics were not needed, since a structurally bracketed corpus was used to extract the training and test sets.

[24] This figure is really an upper bound on their performance for the purposes of comparing to the transformation-based approach, since the transformation-based method *could* be extended to make use of the large amount of extra training material used to obtain this result. Also, proximity effects favorably bias the t-score results when using the larger training corpora. Since many more sentences are being extracted from the same size source corpus, the chance of a test sentence and a training sentence coming from the same paragraph or story is greater, thereby increasing the chance of words seen in the training corpus appearing in the test corpus.

| Method | Accuracy | # of Transforms |
|---|---|---|
| T-Scores | 70.4 - 75.8 | |
| Transformations | 80.8 | 471 |
| Transformations (no N2) | 79.2 | 418 |
| Transformations (classes) | 81.8 | 266 |

Figure 7.12: Comparing Results in PP Attachment.

## 7.4 Why Transformation-Based Learning Works

There are two factors that ultimately contribute to the success or failure of transformation-based error-driven learning: the set of transformations and the set of triggering environments or contexts. The transformations need to be sufficiently powerful to allow improper annotations to be transformed into proper annotations. However, this is not enough. The transformations must be learnable. Learning proceeds by counting: one can observe the effect of carrying out a tranformation in a particular context at a given stage in annotation, and from this observation can learn whether that transformation is effective. Therefore, the system can be successful to the extent that tranformations exist that can convert a poorly annotated sentence into a correctly (or at least better) annotated sentence, and these transformations are sufficiently general that they can be learned. To be learnable, there must exist a set of triggering environments for each transformation that are reliable indicators that a transformation should be applied, that occur frequently, and that can easily be found in the training corpus. Fortunately, it seems as if Zipf's law holds more or less with the transformations learned in the different learning modules described above. In other words, the rank-frequency ratio for transformations is highly skewed. A small number of transformations are extremely effective, meaning they can easily and reliably be observed in the training corpus and will go a long way towards reducing error in the test corpus.

The simplicity of the learner may have a lot to do with its success. To be concrete, take bracketing as an example. Parameters in probabilistic context-free grammars, namely the rules and their probabilities, are all interdependent. At each iteration of the inside-outside algorithm, the entire grammar is considered. In the transformation-based learner, at every stage of learning there is only one thing to find, namely the best transformation at that stage of learning. The learner has a much simpler task at each iteration: finding a single transformation rather than finding a set of rule probabilities where rules are related in complex ways. For example, the learner might learn that at a particular stage of learning a comma is a good indiction of a phrase break. With the inside-outside algorithm, so direct a piece of information could not easily be learned. Since there seem to be many simple and effective cues that can be found within the bounds specified by the transformation templates, the transformation-based learner can easily find them. The same is true for part of speech tagging, where the learner has a mechanism to concisely capture local tagging cues and does not have to resort to brute force recording of statistics.

In addition, since linguistic entities are likely to obey Zipf's law, the entity types seen in the training corpus are likely to account for a large percentage of entity tokens in the test corpus. If, for example, transformations are learned that effectively bracket noun phrases in the training corpus, then a high accuracy rate will likely be achieved on noun phrases in the test corpus.

## 7.5 Conclusions

In this chapter, we have demonstrated that transformation-based error-driven learning can be applied effectively to learn to bracket sentences syntactically. We also demonstrated how the learner can be used to learn how to assign nonterminal labels to an unlabelled syntactic tree. Last, we showed a prepositional phrase postprocesser that could be used to improve parsing accuracy by employing lexical information not used in parsing and by zeroing in on improving parsing accuracy with respect to a particular phenomenon.

# Chapter 8

# Conclusions

In this thesis we have described a new approach to corpus-based language learning, called transformation-based error-driven learning. In this learning paradigm, text is initially naively annotated and then an ordered list of transformations is learned whose application improves annotation accuracy. We have demonstrated that this approach outperforms established statistical approaches in part of speech tagging, text bracketing and prepositional phrase attachment, as well as demonstrating how it could be used to label nonterminal nodes in an unlabelled syntactic tree. This performance is achieved despite the fact that the transformation-based learner is an extremely simple algorithm which is only very weakly statistical[1] and that the structural information that it learns is captured much more succinctly than is typically the case in statistical (and decision-tree) natural language learning systems.

The transformation-based learner can be easily extended simply by adding transformation templates. If a template is not useful, then no transformations will be learned using that particular template. The only possible adverse affect of adding transformations is that it could result in finding a local maximum during learning which blocks the application of other useful transformations, thereby resulting in an overall degradation in performance. One advantage of such a simple system is that a problem like this could be easily detected.

The learner could also be used as a postprocessor to the output of a human annotator or a different automatic annotator simply by changing the start state. Rather than using

[1] Counts are collected and compared, but no more sophisticated statistical relationships are calculated.

the very simple start states used in this thesis, a more sophisticated start state could be used with transformations being learned to patch up weaknesses in the annotation method.

Because the algorithm is extremely simple, development time is very fast. Since absolutely no language-specific or corpus-specific knowledge is hard-coded in the annotation procedure, the annotator is completely portable. In addition, we have shown that with minimal human supervision in the form of a small annotated training corpus, this method can be trained to annotate text with high accuracy.

Once a set of transformations has been learned, the method of applying these transformations is explicitly given. When annotating using transformations, we do not need to search through a set of rules, but rather we simply apply each transformation in order. Transformation-based annotation runs in time linear with respect to the length of the input.

There are a number of advantages to having a very simple algorithm for learning structural information and for effectively applying that information to annotate text. Occam's Razor, originally voiced by William of Occam in 1320, states that a simpler explanation is to be preferred over a more complex one.[2] The learning procedure we have described is simpler than its statistical counterparts in three ways: the learning algorithm itself is simpler, the application algorithm is simpler, and the learned information is stored much more compactly. It is also simple in that there are no hidden parameters or procedures that crucially affect performance and hinder the ability of others to replicate the results and experiment with the learner. In statistical natural language learning, some of the possible hidden factors include: the method for dealing with relative frequencies of zero, handling computer overflow and underflow, threshold values and number of iterations. The complexity of the mathematics behind some of these learning procedures puts them out of reach of many people in the computational linguistics community. The transformation-based approach is very straightforward, simple, and easy to implement, thereby allowing us to concentrate more on the issue of language and less on the issue of statistics. The only potential parameter is the threshold value above which a transformation must score for it to be learned. The performance of the system with respect to this one parameter

[2] This is described a bit more formally in [10].

can easily be observed by learning a set of transformations with the threshold set to zero. Then text can be annotated using this transformation list. For any threshold value, the effect of setting that threshold can be easily observed by measuring performance up to the point where the first transformation is applied that scored below the threshold in training. Therefore, this parameter need not be fixed, but can be set automatically by the computer. Once the start state, transformation templates and scoring function are listed, the learning algorithm and the transformation application algorithm are completely specified. In addition, the transformation-based approach has the added advantage of one algorithm being successful at a number of different structural annotation tasks.

There are a number of exciting future directions in which this work can be continued. It would be interesting to attempt to apply this learning technique to other tasks, such as machine translation and inducing predicate-argument structure. In the tasks we have already attempted, we would like to experiment with different transformation templates and control strategies. We hope this thesis has demonstrated the potential of transformation-based error-driven learning for addressing a wide range of problems in natural language processing.

# Appendix A

# Penn Treebank Part of Speech Tags (Excluding Punctuation)

| | | |
|---|---|---|
| 1. | CC | Coordinating conjunction |
| 2. | CD | Cardinal number |
| 3. | DT | Determiner |
| 4. | EX | Existential "there" |
| 5. | FW | Foreign word |
| 6. | IN | Preposition or subordinating conjunction |
| 7. | JJ | Adjective |
| 8. | JJR | Adjective, comparative |
| 9. | JJS | Adjective, superlative |
| 10. | LS | List item marker |
| 11. | MD | Modal |
| 12. | NN | Noun, singular or mass |
| 13. | NNS | Noun, plural |
| 14. | NP | Proper noun, singular |
| 15. | NPS | Proper noun, plural |
| 16. | PDT | Predeterminer |
| 17. | POS | Possessive ending |
| 18. | PP | Personal pronoun |

| | | |
|---|---|---|
| 19. | PP$ | Possessive pronoun |
| 20. | RB | Adverb |
| 21. | RBR | Adverb, comparative |
| 22. | RBS | Adverb, superlative |
| 23. | RP | Particle |
| 24. | SYM | Symbol |
| 25. | TO | "to" |
| 26. | UH | Interjection |
| 27. | VB | Verb, base form |
| 28. | VBD | Verb, past tense |
| 29. | VBG | Verb, gerund or present participle |
| 30. | VBN | Verb, past participle |
| 31. | VBP | Verb, non-3rd person singular present |
| 32. | VBZ | Verb, 3rd person singular present |
| 33 | WDT | Wh-determiner |
| 34. | WP | Wh-pronoun |
| 35. | WP$ | Possessive wh-pronoun |
| 36. | WRB | Wh-adverb |

# Appendix B

# Old English Part of Speech Tags

| | | |
|---|---|---|
| 1. | AN | Auxiliary Verb (untensed) |
| 2. | AT | Auxiliary Verb (tensed) |
| 3. | CC | Coordinating conjunction |
| 4. | CO | Complementizer |
| 5. | DT | Determiner |
| 6. | JJ | Adjective |
| 7. | NE | Negation |
| 8. | NN | Noun |
| 9. | PDT | Predeterminer |
| 10. | PN | Pronoun |
| 11. | PR | Preposition and subordinating conjunction |
| 12. | RB | Adverb |
| 13. | RP | Particle |
| 14. | TO | To |
| 15. | UH | Exclamation |
| 16. | VBG | Present Participle |
| 17. | VBN | Past Participle |
| 18. | VN | Main Verb (untensed) |
| 19. | VT | Main Verb (tensed) |

# Appendix C

# Original Brown Corpus Tags

| | | |
|---|---|---|
| 1. | . | end of sent |
| 2. | ( | left paren |
| 3. | ) | right paren |
| 4. | – | dash |
| 5. | , | coma |
| 6. | : | colon |
| 7. | “ | open quotes |
| 8. | ” | close quotes |
| 9. | * | not, n't(appended) |
| 10. | abl | pre-qual |
| 11. | abn | pre-quant |
| 12. | abx | pre-quant |
| 13. | ap | post-det |
| 14. | at | art |
| 15. | be | be |
| 16. | bed | were |
| 17. | bedz | was |
| 18. | beg | being |

| | | |
|---|---|---|
| 19. | bem | am |
| 20. | ben | been |
| 21. | ber | are |
| 22. | bez | is |
| 23. | cc | conj |
| 24. | cd | number |
| 25. | cs | subconj |
| 26. | do | do |
| 27. | dod | did |
| 28. | doz | does |
| 29. | dt | sing deter |
| 30. | dti | s/p deter/quant |
| 31. | dts | pl deter |
| 32. | dtx | det/dbl conj |
| 33. | ex | there |
| 34. | hv | have |
| 35. | hvd | had |
| 36. | hvg | having |
| 37. | in | prep |
| 38. | jj | adj |
| 39. | jjr | comp adj |
| 40. | jjs | super adj |
| 41. | jjt | super adj |
| 42. | md | aux |
| 43. | nn | sing noun |
| 44. | nn$ | poss sing noun |
| 45. | nns | pl noun |
| 46. | nns$ | poss pl noun |
| 47. | np | prop noun |
| 48. | np$ | poss prop noun |

| | | |
|---|---|---|
| 49. | nps$ | poss pl prop noun |
| 50. | nr | adv noun |
| 51. | od | ord number |
| 52. | pn | nom pron |
| 53. | pn$ | poss nom pron |
| 54. | pp$ | poss pers pron |
| 55. | pp$$ | sec poss pers pron |
| 56. | ppl | sing per pron |
| 57. | ppls | pl pers pron |
| 58. | ppo | obj pers pron |
| 59. | pps | 3rd sing nom pron |
| 60. | ppss | other nom pron |
| 61. | ql | qual |
| 62. | qlp | post qual |
| 63. | rb | adv |
| 64. | rbr | comp adv |
| 65. | rbt | super adv |
| 66. | rn | nom adv |
| 67. | rp | adv/particle |
| 68. | to | inf |
| 69. | uh | interj |
| 70. | vb | verb |
| 71. | vbd | past verb |
| 72. | vbg | pres part/gerund |
| 73. | vbn | past part |
| 74. | vbz | verb |
| 75. | wdt | wh determ |
| 76. | wp$ | poss wh pron |
| 77. | wpo | obj wh pron |
| 78. | wps | nom wh pron |

| | | |
|---|---|---|
| 79. | wql | wh qual |
| 80. | wrb | wh adv |

# Appendix D

# Penn Treebank Nonterminals

**ADJP**—Adjective phrase. Phrasal category headed by an adjective (including comparative and superlative adjectives). Example: *outrageously expensive*.

**ADVP**—Adverb phrase. Phrasal category headed by an adverb (including comparative and superlative adverbs). Examples: *rather timidly*, *very well indeed*.

**AUX**—Auxiliary Verb Phrase.

**CONJP**—-Coordinate phrase.

**INTJ**—-Interjection

**NEG**—-Negative

**NP**—Noun phrase. Phrasal category that includes all constituents that depend on a head noun.

**PP**—Prepositional phrase. Phrasal category headed by a preposition.

**PRT**—- Particle phrase.

**S**—Simple declarative clause, i.e. one that is not introduced by a (possibly empty) subordinating conjunction or wh-word and that does not exhibit subject-verb inversion.

**SBAR**—Clause introduced by a (possibly empty) subordinating conjunction.

**SBARQ**—Direct question introduced by a wh-word or wh-phrase.

**SINV**—Inverted declarative sentence, i.e. one in which the subject follows the verb.

**SQ**—That part of an SBARQ that excludes the wh-word or wh-phrase.

**VP**—Verb phrase. Phrasal category headed a verb.

**WHADVP**—*Wh*-adverb phrase. Phrasal category headed by a wh-adverb such as *how* or *why*.

**WHNP**—*Wh*-noun phrase. Noun phrase containing (among other things) a wh determiner, as in *which book* or *whose daughter*, or consisting of a wh-pronoun like *who*.

**WHPP**—*Wh*-prepositional phrase. Prepositional phrase containing a wh-determiner, as in *by whatever means necessary*.

**X**—Constituent of unknown or uncertain type.

**?**—A question mark enclosing a constituent (i.e. a question mark preceded by a left parenthesis) means that the parser was unable to decide where to attach the constituent.

# Bibliography


[1] S. Abney. The English noun phrase in its sentential aspects. Unpublished MIT Dissertation, 1987.

[2] L. Bahl, P. Brown, P. DeSouza, and R. Mercer. A tree-based statistical language model for natural language recognition. *Readings in Speech Recognition*, 1990.

[3] J. Baker. Trainable grammars for speech recognition. In *Speech communication papers presented at the 97th Meeting of the Acoustical Society of America*, 1979.

[4] L. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of a Markov process. *Inequalities*, 3:1–8, 1972.

[5] E. Black, S. Abney, D. Flickenger, C. Gdaniec, R. Grishman, P. Harrison, D. Hindle, R. Ingria, F. Jelinek, J. Klavans, M. Liberman, M. Marcus, S. Roukos, B. Santorini, and T. Strzalkowski. A procedure for quantitatively comparing the syntactic coverage of English grammars. In *Proceedings of Fourth DARPA Speech and Natural Language Workshop*, pages 306–311, 1991.

[6] E. Black, F. Jelinek, J. Lafferty, D. Magerman, R. Mercer, and S. Roukos. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993. Columbus, Ohio.

[7] E. Black, F. Jelinek, J. Lafferty, R. Mercer, and S.Roukos. Decision tree models applied to the labeling of text with parts-of-speech. In *Darpa Workshop on Speech and Natural Language*, 1992. Harriman, N.Y.

[8] E. Black, J. Lafferty, and S. Roukos. Development and evaluation of a broad-coverage probabilistic grammar of English-language computer manuals. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, 1992. Newark, De.

[9] L. Bloomfield. *Language*. Holt, New York, 1933.

[10] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Occam's razor. In *Information Processing Letters*, volume 24, 1987.

[11] F. Boas. *Handbook of American Indian Languages, Part 1*. Smithsonian Institution, Washington, D.C., 1911. Bureau of American Ethnology, Bulletin 40.

[12] R. Bod. Using an annotated corpus as a stochastic grammar. In *Proceedings of European ACL*, 1993.

[13] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth and Brooks, 1984.

[14] M. Brent. Automatic acquisition of subcategorization frames from untagged text. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca., 1991.

[15] E. Brill. Discovering the lexical features of a language. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca., 1991.

[16] E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, Trento, Italy, 1992.

[17] E. Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Meeting of the Association of Computational Linguistics*, Columbus, Oh., 1993.

[18] E. Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the ARPA Human Language Technology Workshop*, Princeton, N.J., 1993.

[19] E. Brill. Transformation-based error-driven parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands, 1993.

[20] E. Brill, E. Haeberli, and T. Kroch. Adventures in tagging Old English. Manuscript, 1993.

[21] E. Brill and S. Kapur. An information-theoretic solution to parameter setting. Technical report, Institute for Research in Cognitive Science, University of Pennsylvania Number IRCS-93-07, 1993.

[22] E. Brill, D. Magerman, M. Marcus, and B. Santorini. Deducing linguistic structure from the statistics of large corpora. In *Proceedings of the DARPA Speech and Natural Language Workshop*, pages 275–282, 1990.

[23] E. Brill and M. Marcus. Automatically acquiring phrase structure using distributional analysis. In *Darpa Workshop on Speech and Natural Language*, Harriman, N.Y., 1992.

[24] E. Brill and M. Marcus. Tagging an unfamiliar text with minimal human supervision. In *Proceedings of the Fall Symposium on Probabilistic Approaches to Natural Language*. American Association for Artificial Intelligence (AAAI), 1992.

[25] E. Brill and P. Resnik. A transformation-based approach to prepositional phrase attachment. Technical report, Department of Computer and Information Science, University of Pennsylvania, 1993. Forthcoming.

[26] T. Briscoe and N. Waegner. Robust stochastic parsing using the inside-outside algorithm. In *Workshop notes from the AAAI Statistically-Based NLP Techniques Workshop*, 1992.

[27] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer, and P. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2), 1990.

[28] P. Brown, J. Lai, and R. Mercer. Word-sense disambiguation using statistical methods. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca., 1991.

[29] P. Brown, V. Della Pietra, S. Della Pietra, and R. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 1992.

[30] C. Cardie. Corpus-based acquisition of relative pronoun disambiguation heuristics. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993.

[31] G. Carroll and E. Charniak. Learning probabilistic dependency grammars from labelled text. In *Proceedings of the Fall Symposium on Probabilistic Approaches to Natural Language*. American Association for Artificial Intelligence (AAAI), 1992.

[32] E. Charniak. *A parser with something for everyone.* Academic Press, 1983. In Parsing Natural Language, M. King editor.

[33] S. Chatman. Immediate constituents and expansion analysis. *Word*, 11, 1955.

[34] N. Chomsky. *Aspects of the Theory of Syntax.* MIT Press, Cambridge, 1965.

[35] N. Chomsky. Language and problems of knowledge. Manuscript, 1987.

[36] K. Church. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing, ACL*, 1988.

[37] K. Church, W. Gale, P. Hanks, and D. Hindle. Parsing, word associations and typical predicate-argument relations. In *Proceedings of the International Workshop on Parsing Technologies*, 1989.

[38] T. Cover and J. Thomas. *Elements of Information Theory.* Wiley and Sons, New York, 1991.

[39] D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, Trento, Italy, 1992.

[40] S. Derose. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14, 1988.

[41] G. Ernst and A. Newell. *GPS: A case study in generality and problem solving.* Academic Press, 1969.

[42] R. Quirk et al. *A Comprehensive Grammar of the English Language.* Longman, London, 1985.

[43] S. Fong. *Computational Properties of Principle Based Grammatical Theories.* PhD thesis, Department of Electrical Engineering and Computer Science, MIT, 1991.

[44] W. Francis and H. Kucera. *Frequency analysis of English usage: Lexicon and grammar.* Houghton Mifflin, Boston, 1982.

[45] R. Garside, G. Leech, and G. Sampson. *The Computational Analysis of English: A Corpus-Based Approach.* Longman, London, 1987.

[46] D. Goldberg. *Genetic algorithms in search, optimization, and learning.* Addison-Wesley, 1989.

[47] R. Grishman, L. Hirschman, and N. Nhan. Discovery procedures for sublanguage selectional patterns: Initial experiments. *Computational Linguistics*, 12(3), 1986.

[48] H. Guiter and M. Arapov, editors. *Studies on Zipf's Law.* Studienverlag Dr. N. Brochmeyer, Bochum, 1982. Quantitative Linguistics, Vol. 16.

[49] R. Haigh and G. Sampson adn E. Atwell. Project APRIL - a progress report. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, Buffalo, N.Y., 1988.

[50] J. Hammersley and D. Handscomb. *Monte Carlo Methods.* Chapman and Hall, 1964.

[51] D. Hardt. An algorithm for VP ellipsis. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1992.

[52] Z. Harris. From morpheme to utterance. *Language*, 22, 1946.

[53] Z. Harris. *Structural Linguistics.* University of Chicago Press, Chicago, 1951.

[54] Z. Harris. From phoneme to morpheme. *Language*, 31, 1955.

[55] Z. Harris. *String Analysis of Language Structure.* Mouton and Co., The Hague, 1962.

[56] C. Hemphill, J. Godfrey, and G. Doddington. The ATIS spoken language systems pilot corpus. In *Proceedings of the DARPA Speech and Natural Language Workshop*, 1990.

[57] D. Hindle. Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, 1989.

[58] D. Hindle. Noun classification from predicate-argument structures. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, Pittsburgh, Pa., 1990.

[59] D. Hindle and M. Rooth. Structural ambiguity and lexical relations. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca., 1991.

[60] L. Hirschman, R. Grishman, and N. Sager. Grammatically-based automatic word class formation. *Information Processing and Management*, 11, 1975.

[61] C. Hockett. Review of the mathematical theory of communication by Claude Shannon and Warren Weaver. *Language*, 29, 1953.

[62] P. Holder, editor. *Franz Boas: Introduction to Handbook of American Indian Languages and J.W. Powell: Indian Linguistic Families of America North of Mexico.* University of Nebraska Press, Lincoln, Ne., 1966.

[63] O. Jaeggli and K. Safir, editors. *The null subject parameter.* Foris, Dordrecht, 1989.

[64] F. Jelinek. Continuous speech recognition by statistical methods. *Proc. IEEE*, 64:532–556, 1976.

[65] F. Jelinek. *Impact of Processing Techniques on Communication.* Dordrecht, 1985. In Impact of Processing Techniques on Communication, J. Skwirzinski, ed.

[66] F. Jelinek, J. Lafferty, and R. Mercer. Basic methods of probabilistic context free grammars. Technical report, IBM, Yorktown Heights, 1990. Technical Report RC 16374 (72684).

[67] A. Joshi and L. Levy. Phrase structure trees bear more fruit than you would have thought. *American Journal of Computational Linguistics*, 8(1), 1982.

[68] J. Kimball. Seven principles of surface structure parsing in natural language. *Cognition*, 2, 1973.

[69] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220, 1983.

[70] G. Kiss. Grammatical word classes: A learning process and its simulation. *Psychology of Learning and Motivation*, 7, 1973.

[71] S. Klein and R. Simmons. A computational approach to grammatical coding of English words. *JACM*, 10, 1963.

[72] S. Kullback. *Information Theory and Statistics.* John Wiley and Sons, New York, 1959.

[73] J. Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer speech and language*, 6, 1992.

[74] G. Lakoff. *Women, Fire and Dangerous Things: What Categories Reveal About the Mind.* University of Chicago Press, Chicago, 1987.

[75] K. Lari and S. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4, 1990.

[76] H. Lasnik and M. Saito. On the nature of proper government. *Linguistic Inquiry*, 15(2), 1984.

[77] L. Levy and A. Joshi. Skeletal structural descriptions. *Information and Control*, 39(2), 1978.

[78] B. MacWhinney and C. Snow. The child language data exchange system. *Journal of Child Language*, 12, 1985.

[79] D. Magerman and M. Marcus. Parsing a natural language using mutual information statistics. In *Proceedings, Eighth National Conference on Artificial Intelligence (AAAI 90)*, 1990.

[80] B. Mandelbrot. *An information theory of the statistical structure of language*. London, 1953. In Communication Theory, W. Jackson, ed.

[81] M. Marcus. *A theory of syntactic recognition for natural language*. MIT Press, 1980.

[82] M. Marcus. *Some Inadequate Theories of Human Language Processing*. 1984. In: Talking Minds: The Study of Language in the Cognitive Sciences, Bever, T., Carroll, J. and Miller, L. eds.

[83] M. Marcus, B. Santorini, and M. Marcinkiewicz. Building a large annotated corpus of English: the Penn Treebank. To appear in Computational Linguistics, 1993.

[84] M. Meteer, R. Schwartz, and R. Weischedel. Empirical studies in part of speech labelling. In *Proceedings of the fourth DARPA Workshop on Speech and Natural Language*, 1991.

[85] G. Miller. Wordnet: an on-line lexical database. *International Journal of Lexicography*, 1990.

[86] A. Newell and H. Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[87] M. Niv. Resolution of syntactic ambiguity: the case of new subjects. In *Proceedings of the 15th Annual Meeting of the Cognitive Science Society*, 1993.

[88] F. Pereira and Y. Schabes. Inside-outside reestimation from partially bracketed corpora. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, Newark, De., 1992.

[89] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, 1993.

[90] S. Pinker. *Learnability and Cognition.* MIT Press, Cambridge, 1989.

[91] J. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[92] J. Quinlan and R. Rivest. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80, 1989.

[93] L. Rabiner. A tutorial on Hidden Markov models and selected applications in speech recognition. In *Readings in speech recognition*, 1990. A Waibel and K. Lee, Editors.

[94] P. Resnik. Semantic classes and syntactic ambiguity. In *ARPA Workshop on Human Language Technology*, 1993.

[95] R. Rivest. Learning decision lists. *Machine Learning*, 2, 1987.

[96] R. Robins. Noun and verb in universal grammar. *Language*, 28(3), 1953.

[97] A. Rosenfeld, H. Huang, and V. Schneider. An application of cluster detection to text and picture processing. *IEEE Transactions on Information Theory*, 15(6), 1969.

[98] A. Rouvret and J. Vergnaud. Specifying reference to the subject. *Linguistic Inquiry*, 11(1), 1980.

[99] G. Sampson. *Schools of Linguistics.* Stanford University Press, 1980.

[100] G. Sampson. A stochastic approach to parsing. In *Proceedings of COLING 1986*, Bonn, 1986.

[101] E. Sapir. *Language.* New York, 1921.

[102] Y. Schabes, M. Roth, and R. Osborne. Parsing the Wall Street Journal with the inside-outside algorithm. In *Proceedings of the 1993 European ACL*, Uterich, The Netherlands, 1993.

[103] S. Seneff. Tina: A natural language system for spoken language applications. *Computational Linguistics*, 1992.

[104] R. Sharman, F. Jelinek, and R. Mercer. Generating a grammar for statistical training. In *Proceedings of the 1990 Darpa Speech and Natural Language Workshop*, 1990.

[105] H. Simon. On a class of skew distribution functions. *Biometrika*, 42, 1955.

[106] W. Stolz. A probabilistic procedure for grouping words into phrases. *Language and Speech*, 8, 1965.

[107] A. Taylor and T. Kroch. The Penn parsed corpus of Middle English: a syntactically annotated database. Presented at the Georgetown University Roundtable on Languages and Linguistics Pre-session on Corpus-Based Linguistics, 1993.

[108] R. Thomason, editor. *Formal Philosophy: Selected Papers of Richard Montague*. Yale University Press, 1974.

[109] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Inform. Theory*, IT-13, 1967.

[110] L. Wall and R. Schwartz. *Programming perl*. O'Reilly and Associates, 1991.

[111] R. Wells. Immediate constituents. *Language*, 23, 1947.

[112] B. Whorf. *Language, Thought and Reality: Selected Writings of Benjamin Lee Whorf*. MIT Press, Cambridge, 1956.

[113] T. Wonnacott and R. Wonnacott. *Introductory Statistics for Business and Economics*. Wiley, 1984.

[114] G. Zipf. *Selected Studies of the Principle of Relative Frequency in Language*. Harvard University Press, 1932.

[115] G. Zipf. *The Psycho-Biology of Language*. Houghton Mifflin, 1935.

[116] G. Zipf. *Human behavior and the principle of least effort*. Hafner, New York, 1949.