Machine Learning, 34, 151–178 (1999)

Learning to Parse Natural Language with Maximum Entropy Models

ADWAIT RATNAPARKHI* adwait@unagi.cis.upenn.edu
Dept. of Computer and Information Science
University of Pennsylvania
200 South 33rd Street
Philadelphia, PA 19104-6389

Abstract. This paper presents a machine learning system for parsing natural language that learns from manually parsed example sentences, and parses unseen data at state-of-the-art accuracies. Its machine learning technology, based on the maximum entropy framework, is highly reusable and not specific to the parsing problem, while the linguistic hints that it uses to learn can be specified concisely. It therefore requires a minimal amount of human effort and linguistic knowledge for its construction. In practice, the running time of the parser on a test sentence is linear with respect to the sentence length. We also demonstrate that the parser can train from other domains without modification to the modeling framework or the linguistic hints it uses to learn. Furthermore, this paper shows that research into rescoring the top 20 parses returned by the parser might yield accuracies dramatically higher than the state-of-the-art.

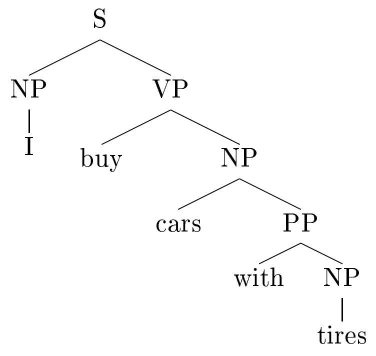
Keywords: maximum entropy models, natural language processing, parsing

1. Introduction

The task of a natural language parser is to take a sentence as input and return a syntactic representation that corresponds to the likely semantic interpretation of the sentence. For example, some parsers, given the sentence

I buy cars with tires

would return a parse tree in the format:



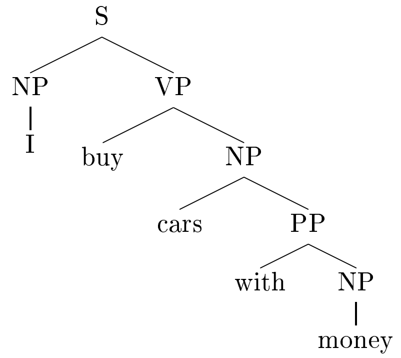
* The author is now working at the IBM TJ Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

where the non-terminal labels denote the type of phrase (e.g., "PP" stands for prepositional phrase). Accurate parsing is difficult because subtle aspects of word meaning—from the parser's view—dramatically affect the interpretation of the sentence. For example, given the sentence

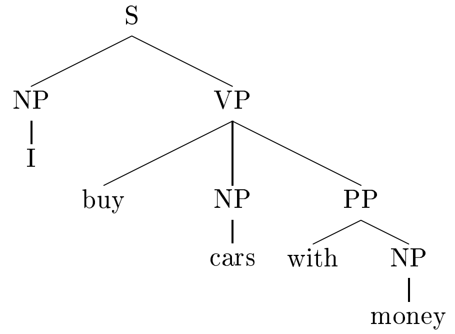
I buy cars with money

a parser might propose the following two parses

- (Unlikely:)



- (Likely:)



Both parses are grammatical, in the sense that a typical context free grammar for English will allow both structures, but only one corresponds to the likely interpretation of the sentence. A parser actually needs detailed semantic knowledge of certain key words in the sentence in order to distinguish the likely parse; it needs to somehow know that *with money* refers to *buy* and not *cars*.

The parsers which currently show superior accuracies on freely occurring text are all classified as *statistical* or *corpus-based*, since they automatically learn to approximate syntactic and semantic knowledge for parsing from a large corpus of text, called a *treebank*, that has been manually annotated with syntactic information. In order to evaluate the accuracy of a statistical parser, we first train it on a subset of the treebank, test it on another non-overlapping subset, and then compare the labelled syntactic constituents it proposes with the labelled syntactic constituents in the annotation of the treebank. The labelled constituent accuracies of the best parsers approach roughly 90% when tested on freely occurring sentences in the Wall St. Journal domain.

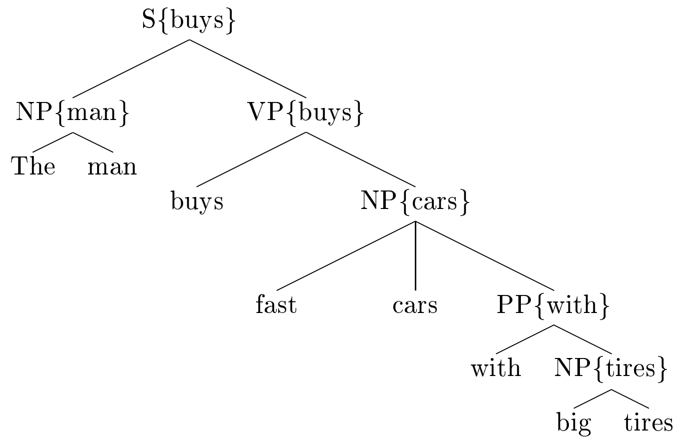


Figure 1. A parse tree annotated with head words

2. Previous Work

Recent corpus-based parsers differ in the simplicity of their representation and the degree of supervision necessary, but agree in that they resolve parse structure ambiguities by looking at certain cooccurrences of constituent *head words* in the ambiguous parse. A head word of a constituent, informally, is the one word that best represents the meaning of the constituent, e.g., Figure 1 shows a parse tree annotated with head words. Parsers vary greatly on how head word information is used to disambiguate possible parses for an input sentence. Black et al. (1993) introduces *history-based* parsing, in which decision tree probability models, trained from a treebank, are used to score the different derivations of sentences produced by a hand-written grammar. Jelinek et al. (1994), Magerman (1995) also train history-based decision tree models from a treebank for use in a parser, but do not require an explicit hand-written grammar. These decision trees do not look at words directly, but instead represent words as bitstrings derived from an automatic clustering technique. In contrast, Hermjakob and Mooney (1997) use a rich semantic representation when training decision tree and decision list techniques to drive parser actions.

Several other recent parsers use statistics of pairs of head words in conjunction with chart parsing techniques to achieve high accuracy. Collins (1996, 1997) uses chart-parsing techniques with head word bigram statistics derived from a treebank. Charniak (1997) uses head word bigram statistics with a probabilistic context free grammar, while Goodman (1997) uses head word bigram statistics with a probabilistic feature grammar. Collins (1996), Goodman (1997), Charniak (1997), Collins (1997) do not use general machine learning algorithms, but instead develop specialized statistical estimation techniques for their respective parsing tasks.

The parser in this paper attempts to combine the advantages of other approaches. It uses a natural and direct representation of words in conjunction with a general

Table 1. Tree-Building Procedures of Parser

Pass	Procedure	Actions	Description
First Pass	TAG	A POS tag in tag set	Assign POS Tag to word
Second Pass	CHUNK	`Start X`, `Join X`, `Other`	Assign Chunk tag to POS tag and word
Third Pass	BUILD	`Start X`, `Join X`, where `X` is a constituent label in label set	Assign current tree to start a new constituent, or to join the previous one
	CHECK	`Yes`, `No`	Decide if current constituent is complete

machine learning technique, maximum entropy modeling. We argue that the successful use of a simple representation with a general learning technique is the combination that both *minimizes human effort* and maintains state-of-the-art parsing accuracy.

3. Parsing with Maximum Entropy Models

The parser presented here constructs labelled syntactic parse trees with actions similar to those of a standard *shift–reduce* parser. (Many other parsing techniques exist for natural language, see Allen (1995) for an introduction.) The sequence of actions $\{a_1 \ldots a_n\}$ that construct a completed parse tree T are called the *derivation* of T. There is no explicit grammar that dictates what actions are allowable; instead, all actions that lead to a well-formed parse tree are allowable and maximum entropy probability models are used to score each action. The maximum entropy models are trained by examining the derivations of the parse trees in a treebank. The individual scores of the actions in a derivation are used to compute a score for the whole derivation, and hence the whole parse tree. When parsing a sentence, the parser uses a search procedure that efficiently explores the space of possible parse trees, and attempts to find the highest scoring parse tree.

Section 3.1 describes the actions of the tree-building procedures, section 3.2 describe the maximum entropy probability models, and section 3.3 describes the algorithm that searches for the highest scoring parse tree.

3.1. Actions of the Parser

The actions of the parser are produced by *procedures*, that each take a derivation $d = \{a_1 \ldots a_n\}$, and predict some action a_{n+1} to create a new derivation $d' = \{a_1 \ldots a_{n+1}\}$. The actions of the procedures are designed so that any possible complete parse tree T has *exactly one* derivation.

The procedures are called TAG, CHUNK, BUILD, and CHECK, and are applied in three left-to-right passes over the input sentence; the first pass applies TAG, the second pass applies CHUNK, and the third pass applies BUILD and CHECK. The passes, the procedures they apply, and the actions of the procedures are summarized in Table 1. Typically, the parser explores many different derivations when parsing a sentence, but for illustration purposes, Figures 2–8 trace one possible derivation for the sentence "I saw the man with the telescope", using the consituent labels and part-of-speech tags of the University of Pennsylvania treebank (Marcus et al., 1994).

The actions of the procedures are scored with maximum entropy probability models that use information in the local context to compute their probabilities. (A more detailed discussion of the probability models will occur in Section 3.2.) Using three passes instead of one pass allows the the use of more local context. For example, the model for the CHUNK procedure will have the output from TAG in its left and right context, and the models for the BUILD and CHECK procedures will have the output of TAG and CHUNK and their left and right contexts. If all these procedures were implemented in one left-to-right pass, the model for CHUNK would not have the output of TAG in its right context, and the models for BUILD and CHECK would not have the output of TAG and CHUNK in their right context.

3.1.1. First Pass The first pass takes an input sentence, shown in Figure 2, and uses TAG to assign each word a part-of-speech (POS) tag. The result of applying TAG to each word is shown in Figure 3. The tagging phase resembles other standalone statistical taggers in the literature (Weischedel et al., 1993) but is integrated into the parser's search procedure, so that the parser does not need to commit to a single POS tag sequence.

3.1.2. Second Pass The second pass takes the output of the first pass and uses CHUNK to determine the "flat" phrase chunks of the sentence, where a phrase is "flat" if and only if it is a constituent whose children are not constituents. Starting from the left, CHUNK assigns each (word,POS tag) pair a "chunk" tag, either `Start X`, `Join X`, or `Other`, where `X` is a constituent label. Figure 4 shows the result after the second pass. The chunk tags are then used for chunk detection, in which any consecutive sequence of words $w_m \dots w_n$ $(m \leq n)$ are grouped into a "flat" chunk X if w_m has been assigned `Start X` and $w_{m+1} \dots w_n$ have all been assigned `Join X`. The result of chunk detection, shown in Figure 5, is a forest of trees and serves as the input to the third pass.

The granularity of the chunks, as well as the possible constituent labels of the chunks, are determined from the treebank that is used to train the parser. Examples of constituents that are marked as flat chunks in the Wall St. Journal domain of the Penn treebank include noun phrases such as *a nonexecutive director*, adjective phrases such as *61 years old*, and quantifier phrases such as *about $ 370 million*.

The chunking in our second pass differs from other chunkers in the literature (Ramshaw and Marcus, 1995; Church, 1988) in that it finds chunks of all constituent

Table 2. Comparison of BUILD and CHECK to operations of a shift–reduce parser

Procedure	Actions	Similar Shift–Reduce Parser Action
CHECK	`No`	shift
CHECK	`Yes`	reduce α, where α is CFG rule of proposed constituent
BUILD	`Start X`, `Join X`	Determines α for subsequent reduce operations

I saw the man with the telescope

Figure 2. Initial Sentence

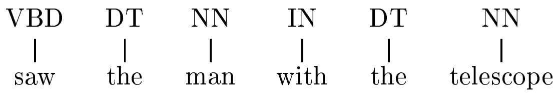


Figure 3. The result after First Pass

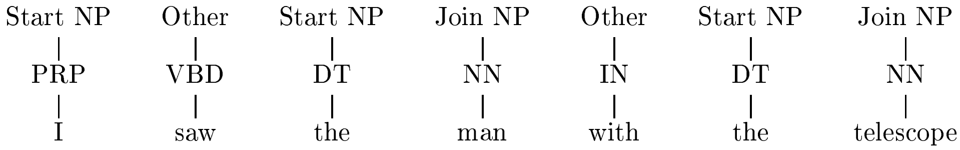


Figure 4. The result after Second Pass

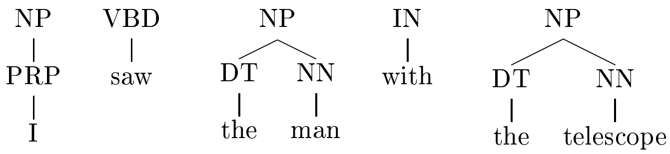


Figure 5. The result of chunk detection

labels, and not just noun phrase chunks. Our multi-pass approach is similar to the approach of the parser in Abney (1991), which also first finds chunks in one pass, and then attaches them together in the next pass.

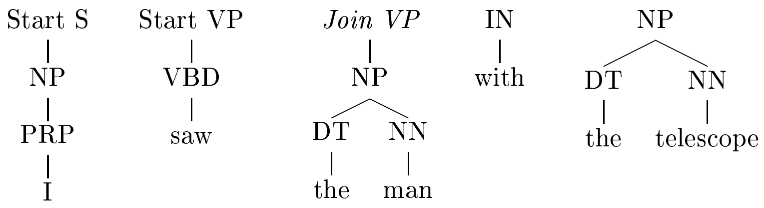


Figure 6. An application of BUILD in which `Join VP` is the action

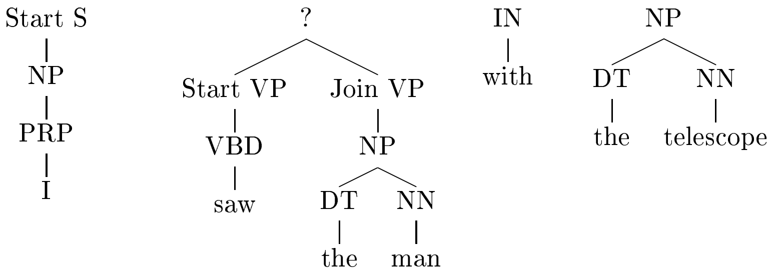


Figure 7. The most recently proposed constituent (shown under ?)

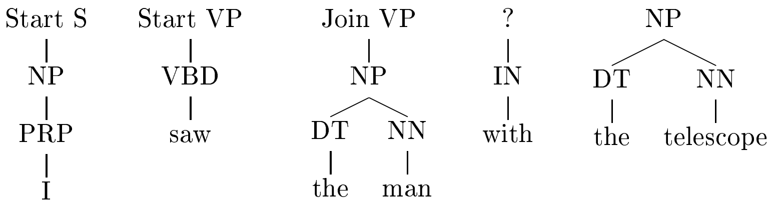


Figure 8. An application of CHECK in which `No` is the action, indicating that the proposed constituent in figure 7 is *not* complete. BUILD will now process the tree marked with ?. The derivation of this partially completed tree is { PRP, VBD, DT, NN, IN, DT, NN, Start NP, Other, Start NP, Join NP, Other, Start NP, Join NP, Start S, no, Start VP, no, Join VP, no }.

3.1.3. Third Pass The third pass always alternates between the use of BUILD and CHECK, and completes any remaining constituent structure. BUILD decides whether a tree will start a new constituent or join the incomplete constituent immediately to its left. Accordingly, it annotates the tree with either `Start X`, where `X` is any constituent label, or with `Join X`, where `X` matches the label of the incomplete constituent to the left. BUILD always processes the leftmost tree without any `Start X` or `Join X` annotation. Figure 6 shows an application of BUILD in which the action is `Join VP`. After BUILD, control passes to CHECK, which finds the most recently proposed constituent, and decides if it is complete. The most recently proposed constituent, shown in Figure 7, is the rightmost sequence of trees $t_m \dots t_n$ $(m \leq n)$ such that t_m is annotated with `Start X` and $t_{m+1} \dots t_n$ are annotated with `Join X`. If CHECK decides `yes`, then the proposed constituent takes its place in the forest as an actual constituent, on which BUILD does its work. Otherwise, the constituent is not finished and BUILD processes the next tree in the forest, t_{n+1}. We force CHECK to answer `no` if the proposed constituent is a "flat" chunk, since such constituents must be formed in the second pass. (Otherwise, flat chunks would not have unique derivations.) Figure 8 shows the result when CHECK looks at the proposed constituent in Figure 7 and decides `No`. The third pass terminates when CHECK is presented with a constituent that spans the entire sentence.

A complete derivation for an n word sentence consists of n actions of TAG, n actions of CHUNK, and alternating actions of BUILD and CHECK. For reference purposes, the derivation of the partially completed tree in Figure 8 is included in the caption. The constituent labels produced by BUILD, i.e., the types of `X` in the `Start X` and `Join X` actions, are determined from the treebank that is used to train the parser.

Table 2 compares the actions of BUILD and CHECK to the operations of a standard shift–reduce parser. The `No` and `Yes` actions of CHECK correspond to the shift and reduce actions, respectively. The important difference is that while a shift–reduce parser creates a constituent in one step (reduce α), the procedures BUILD and CHECK create it over several steps in smaller increments.

While the use of maximum entropy models together with shift–reduce parsing is novel (to our knowledge), shift–reduce parsing techniques have been popular in the natural language literature. Aho et al. (1988) describe shift–reduce parsing techniques (for programming languages) in detail, Marcus (1980) uses shift–reduce parsing techniques for natural language, and Briscoe and Carroll (1993) describe probabilistic approaches to *LR parsing*, a type of shift–reduce parsing. Other recent machine learning approaches to shift–reduce parsing include Magerman (1995) and Hermjakob and Mooney (1997).

3.2. Probability Models that Use Context to Predict Parsing Actions

The parser uses a *history-based* approach (Black et al., 1993), in which a probability $p_X(a|b)$ is used to score an action a of procedure $X \in \{$ TAG, CHUNK, BUILD, CHECK $\}$, depending on the partial derivation b (also called a *context* or *history*) that is available at the time of the decision. The conditional probability models

p_X are estimated under the maximum entropy framework. The advantage of this framework is that we can use arbitrarily diverse information in the context b when computing the probability of an action a of some procedure X.

While any context b is a rich source of information, it is difficult to know *exactly* what information is useful for parsing. However, we would like to implement the following inexact intuitions about parsing:

- Using constituent head words is useful.
- Using combinations of head words is useful.
- Using less-specific information is useful.
- Allowing limited lookahead is useful.

The above intuitions are implemented in the maximum entropy framework as *features*, and each feature is assigned a "weight" which corresponds to how useful it is for modeling the data. We will later show that a mere handful of guidelines are sufficient to completely describe the feature sets used by the parsing models.

3.2.1. The Maximum Entropy Framework The maximum entropy framework is a clean way for experimenters to combine evidence thought to be useful in modeling data. While the exact nature of the evidence is task dependent, the framework itself is independent of the parsing task and can be used for many other problems, like language modeling for speech recognition (Lau et al., 1993; Rosenfeld, 1996) and machine translation (Berger et al., 1996). The basic unit of evidence in this framework is a *feature*, a function f:

$$f : \mathcal{A} \times \mathcal{B} \rightarrow \{0, 1\} \tag{1}$$

where $\mathcal{A}$ is the set of possible actions, and $\mathcal{B}$ is the set of possible contexts. A feature, given an (a, b) pair, captures any information in b that might be useful for predicting a. Given a training set $\mathcal{T} = \{(a_1, b_1), \ldots, (a_n, b_n)\}$, define $\tilde{p}(a, b)$ as the observed probability of the pair (a, b) in the training set, and define $E_{\tilde{p}} f_j$ as the observed expectation of feature f_j:

$$E_{\tilde{p}} f_j = \sum_{a,b} \tilde{p}(a, b) f_j(a, b)$$

Intuitively, $E_{\tilde{p}} f_j$ is just the normalized count of the feature f_j in the training set $\mathcal{T}$. (We will later describe in Section 3.2.3 how to obtain training sets from a treebank.) We desire a conditional probability model p^* that is consistent with the observed expectation of f_j, but also one that is likely to generalize well to unseen data. The Principle of Maximum Entropy (Jaynes, 1957) recommends that we choose the model p^* with the highest entropy over the set of those models that are consistent with the observed expectations, i.e., the model is that is maximally noncomittal beyond meeting the observed evidence. We follow the conditional maximum entropy framework described in Berger et al. (1996), which chooses p^* such that

$$\begin{aligned}
p^* &= \operatorname*{argmax}_{p \in P} H(p) \\
P &= \{p | E_p f_j = E_{\tilde{p}} f_j \quad j = 1 \ldots k\} \\
E_p f_j &= \sum_{a,b} \tilde{p}(b) p(a|b) f_j(a,b) \\
H(p) &= -\sum_{a,b} \tilde{p}(b) p(a|b) \log p(a|b)
\end{aligned}$$

where $f_1 \ldots f_k$ are the features in the model, $\tilde{p}(b)$ is the observed probability of a context b in the training set, P is the set of consistent models, $E_p f_j$ is the model's expectation of f_j, and $H(p)$ is the entropy of the model p, averaged over the contexts of the training set. The form of the solution for p^* is

$$p^*(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)} \tag{2}$$

$$Z(b) = \sum_{a \in \mathcal{A}} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}$$

where $\alpha_1 \ldots \alpha_k$ are the parameters of the model ($\alpha_j > 0$), and $Z(b)$ is a normalization factor.

There is an interesting relationship between maximum likelihood estimates of models of form (2) and maximum entropy models. It also the case that:

$$\begin{aligned}
p^* &= \operatorname*{argmax}_{q \in Q} L(q) \\
Q &= \{p \mid p(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{k} \alpha_j^{f_j(a,b)}\} \\
L(p) &= \sum_{(a,b)} \tilde{p}(a,b) \log p(a|b)
\end{aligned}$$

where Q is the set of models of form (2), and where $L(p)$ is proportional to the log-likelihood of the training set according to the model p. Therefore $p^* = \operatorname{argmax}_{q \in Q} L(q) = \operatorname{argmax}_{p \in P} H(p)$ and p^* can be viewed under both the maximum entropy and maximum likelihood frameworks: it maximizes the entropy over the set of consistent models P and maximizes likelihood over the set of models of form (2), Q. The duality is appealing since as a maximum entropy model, p^* will not assume anything beyond the evidence, and as a maximum likelihood model, p^* will have a close fit to the observed data. The maximum entropy framework and its duality with maximum likelihood estimation are discussed in more detail elsewhere (Berger et al., 1996; Della Pietra et al., 1997).

An advantage of this framework is that there are no independence assumptions or inherent restrictions on the features beyond the form (1). Therefore, experimenters can add arbitrarily diverse or complicated features to the parsing models. This advantage is significant because informative features in parsing (described below in section 3.2.2) are often inter-dependent by nature.

3.2.2. Features All evidence in the maximum entropy framework must be expressed through features, and any feature is implemented with a function $cp : \mathcal{B} \rightarrow \{true, false\}$, called a *contextual predicate.* A contextual predicate checks for the presence or absence of useful information in a context $b \in \mathcal{B}$ and returns true or false accordingly. In this implementation of the maximum entropy framework, every feature f has the format

$$f(a,b) = \begin{cases} 1 & \text{if } cp(b) = \texttt{true} \;\&\&\; a = a' \\ 0 & \text{otherwise} \end{cases}$$

and therefore expresses a cooccurrence relationship between some action a' and some linguistic fact about the context captured by cp.

The contextual predicates for a procedure X are denoted by $\mathcal{CP}_X$, and Table 3 specifies the guidelines, or templates, for creating $\mathcal{CP}_X$, where $X \in \{$ TAG, CHUNK, BUILD, CHECK $\}$. The templates are only linguistic hints, in that they do not specify the information itself, but instead, specify *the location* of the useful information in a context b. The templates use indices relative to the tree that is currently being modified. For example, if the current tree is the 5th tree, $\mathsf{cons}(-2)$ looks at the constituent label, head word, and start/join annotation of the 3rd tree in the forest. The actual contextual predicates in $\mathcal{CP}_X$ are obtained automatically, by recording certain aspects of the context (specified by the templates) in which procedure X was used in the derivations of the trees in the treebank. For an example, an actual contextual predicate $cp \in \mathcal{CP}_{\text{BUILD}}$, derived (automatically) from the template $\mathsf{cons}(0)$, might be

$$cp(b) = \begin{cases} true & \text{if the 0th tree of } b \text{ has label “NP” and head word “he”} \\ false & \text{otherwise} \end{cases}$$

In order to obtain this predicate, there must exist a derivation in the manually parsed example sentences in which BUILD decides an action in the presence of some partial derivation b, such that the 0th tree of b had a constituent label “NP” and head word “he”. Constituent head words are found, when necessary, with the algorithm in Black et al. (1993), Magerman (1995).

Contextual predicates which look at head words, or especially pairs of head words, may not be reliable predictors for the procedure actions due to their sparseness in the training set. Therefore, for each lexically based contextual predicate, there also exist one or more corresponding less specific contextual predicates which look at the same context, but *omit* one or more words. For example, the templates $\mathsf{cons}(0, 1^*)$, $\mathsf{cons}(0^*, 1)$, $\mathsf{cons}(0^*, 1^*)$ are the same as $\mathsf{cons}(0, 1)$ but omit references to the head word of the 1st tree, the 0th tree, and both the 0th and 1st tree, respectively.

The less specific contextual predicates should allow the model to provide reliable probability estimates when the words in the history are rare. Less specific predicates are not enumerated in Table 3, but their existence is indicated with a * and †. The *default* predicates in Table 3 return true for any context and are the least specific (and most frequent) predicates; they should provide reasonable estimates when the model encounters a context in which every other contextual predicate is unreliable.

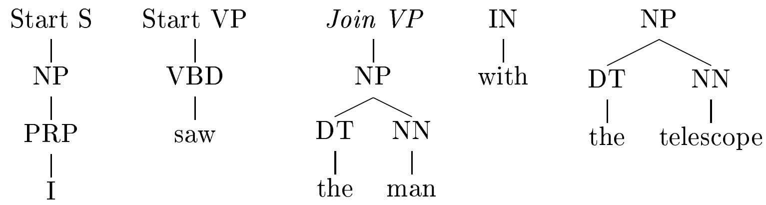
The contextual predicates attempt to capture the intuitions about parsing information discussed earlier. For example, predicates derived from templates like $\mathsf{cons}(0)$ look at constituent head words, while predicates derived from templates like $\mathsf{cons}(-1, 0)$ look at combinations of head words. Predicates derived from templates like $\mathsf{cons}(-1^*, 0)$ look at less specific information, while predicates derived from templates like $\mathsf{cons}(0, 1, 2)$ use limited lookahead. Furthermore, the information expressed in the predicates is always local to where the parsing action is taking place. The contextual predicates for TAG, discussed elsewhere (Ratnaparkhi, 1996), look at the previous 2 words and tags, the current word, and the following 2 words. The contextual predicates for CHUNK look at the previous 2 words, tags, and chunk labels, as well as the current and following 2 words and tags. BUILD uses head word information from the previous 2 and current trees, as well as the following 2 chunks, while CHECK looks at the surrounding 2 words and the head words of the children of the proposed constituent. The intuitions behind the contextual predicates are not linguistically deep, and as a result, the information necessary for parsing can be specified concisely with only a few templates.

3.2.3. Training Events The contextual predicates for a procedure X are used to encode the derivations in the treebank as a set of training events $\mathcal{T}_X = \{ (a_1, b_1), \ldots, (a_N, b_N) \}$. Each $(a, b) \in \mathcal{T}_X$ represents an action of procedure X in a derivation and is encoded as $(a, cp_1 \ldots cp_k)$, where $cp_1 \ldots cp_k$ are contextual predicates such that $cp_i \in \mathcal{CP}_X$ and $cp_i(b) = true$, for $1 \leq i \leq k$, and where b is the context in which action a occurred for procedure X. For example, Figure 9 shows the encoding of a partial derivation in which the BUILD procedure predicts `Join VP`. While any context $b \in \mathcal{B}$ is, in practice, encoded as a sequence of contextual predicates, the encoding is just an implementation choice; the mathematics of the maximum entropy framework do not rely upon any one particular encoding of the space of possible contexts $\mathcal{B}$. The training events $\mathcal{T}_X$ for a procedure $X \in \{$ TAG, CHUNK, BUILD, CHECK $\}$ are used for feature selection and parameter estimation, described below.

3.2.4. Feature Selection Feature selection refers to the process of choosing a useful subset of features $\mathcal{S}_X$ from the set of all possible features $\mathcal{P}_X$ for use in the maximum entropy model corresponding to procedure X. If $\mathcal{CP}_X$ are all the contextual predicates used to encode the training events $\mathcal{T}_X$, and $\mathcal{A}_X$ are the possible

Table 3. Contextual Information Used by Probability Models (* = all possible less specific contexts are used, † = if a less specific context includes a word, it must include the head word of the current tree, i.e., the 0th tree.)

Procedure	Templates	Description	Templates Used
TAG		See Ratnaparkhi (1996)	
CHUNK	chunkandpostag(n)*	The word, POS tag, and chunk tag of nth leaf. Chunk tag omitted if $n \geq 0$.	chunkandpostag(0), chunkandpostag(-1), chunkandpostag(-2) chunkandpostag(1), chunkandpostag(2)
	chunkandpostag(m, n)*	chunkandpostag(m) & chunkandpostag(n)	chunkandpostag($-1, 0$), chunkandpostag($0, 1$)
	default	Returns true for any context.	
BUILD	cons(n)	The head word, constituent (or POS) label, and start/join annotation of the nth tree. Start/join annotation omitted if $n \geq 0$.	cons(0), cons(-1), cons(-2), cons(1), cons(2)
	cons(m, n)*	cons(m) & cons(n)	cons($-1, 0$), cons($0, 1$)
	cons(m, n, p)†	cons(m), cons(n), & cons(p).	cons($0, -1, -2$), cons($0, 1, 2$), cons($-1, 0, 1$)
	punctuation	The constituent we could join (1) contains a "[" and the current tree is a "]"; (2) contains a "," and the current tree is a ","; (3) spans the entire sentence and current tree is "."	bracketsmatch, iscomma, endofsentence
	default	Returns true for any context.	
CHECK	checkcons(n)*	The head word, constituent (or POS) label of the nth tree, and the label of proposed constituent. *begin* and *last* are first and last child (resp.) of proposed constituent.	checkcons($last$), checkcons($begin$)
	checkcons(m, n)*	checkcons(m) & checkcons(n)	checkcons($i, last$) $begin \leq i < last$
	production	Constituent label of parent (X), and constituent or POS labels of children ($X_1 \ldots X_n$) of proposed constituent	production=$X \rightarrow X_1 \ldots X_n$
	surround(n)*	POS tag and word of the nth leaf to the left of the constituent, if $n < 0$, or to the right of the constituent, if $n > 0$	surround(1), surround(2), surround(-1), surround(-2)
	default	Returns true for any context.	



The above action (Join VP) is encoded as follows (a vertical bar | separates information from the same subtree, while a comma , separates information from different subtrees. A tilde ~ denotes a constituent label, as opposed to a part-of-speech tag.):

```
Action = JoinVP

Contextual Predicates =
DEFAULT
cons(0)=~NP|man
cons(0*)=~NP
cons(-1)=StartVP|VBD|saw
cons(-1*)=StartVP|VBD
cons(-2)=StartS|~NP|I
cons(-2*)=StartS|~NP
cons(1)=IN|with
cons(1*)=IN
cons(2)=~NP|telescope
cons(2*)=~NP
cons(-1*,0*)=StartVP|VBD,~NP
cons(-1,0*)=StartVP|VBD|saw,~NP
cons(-1*,0)=StartVP|VBD,~NP|man
cons(-1,0)=StartVP|VBD|saw,~NP|man
cons(0*,1*)=~NP,IN
cons(0,1*)=~NP|man,IN
cons(0*,1)=~NP,IN|with
cons(0,1)=~NP|man,IN|with
cons(0*,1*,2*)=~NP,IN,~NP
cons(0,1*,2*)=~NP|man,IN,~NP
cons(0,1,2*)=~NP|man,IN|with,~NP
cons(0,1*,2)=~NP|man,IN,~NP|telescope
cons(0,1,2)=~NP|man,IN|with,~NP|telescope
cons(-1*,0*,1*)=StartVP|VBD,~NP,IN
cons(-1*,0,1*)=StartVP|VBD,~NP|man,IN
cons(-1,0,1*)=StartVP|VBD|saw,~NP|man,IN
cons(-1*,0,1)=StartVP|VBD,~NP|man,IN|with
cons(-1,0,1)=StartVP|VBD|saw,~NP|man,IN|with
cons(-2*,-1*,0*)=StartS|~NP,StartVP|VBD,~NP
cons(-2*,-1*,0)=StartS|~NP,StartVP|VBD,~NP|man
cons(-2,-1*,0)=StartS|~NP|I,StartVP|VBD,~NP|man
cons(-2*,-1,0)=StartS|~NP,StartVP|VBD|saw,~NP|man
cons(-2,-1,0)=StartS|~NP|I,StartVP|VBD|saw,~NP|man
```

Figure 9. Encoding a derivation with contextual predicates

actions for procedure X, the set of possible features $\mathcal{P}_X$ for use in X's model are:

$$\begin{aligned}\mathcal{P}_X &= \{f \mid f(a,b) = \begin{cases} 1 & \text{if } cp(b) = \texttt{true} \;\&\&\; a = a' \\ 0 & \text{otherwise} \end{cases} \\ &\qquad \text{where } cp \in \mathcal{CP}_X \text{ and } a' \in \mathcal{A}_X\}\end{aligned}$$

Thus any contextual predicate cp that occurs with any action a' can potentially be a feature. However, many of these features occur infrequently, and are therefore not reliable sources of evidence since their behavior in the training events may not represent their behavior in unseen data. For example, it is unlikely that all of the contextual predicates in Table 9 would form reliable features.

We use a very simple feature selection strategy: assume that any feature that occurs less than 5 times is noisy and discard it. Feature selection with a count cutoff does not yield a *minimal* feature set; many of the selected features will be redundant. However, in practice, it yields a feature set that is mostly noise-free with almost no computational expense. Therefore, the selected features for use in procedure X's model are

$$\begin{aligned}\mathcal{S}_X &= \{f \mid f(a,b) = \begin{cases} 1 & \text{if } cp(b) = \texttt{true} \;\&\&\; a = a' \\ 0 & \text{otherwise} \end{cases} \\ &\qquad \text{where } cp \in \mathcal{CP}_X \text{ and } a' \in \mathcal{A}_X, \sum_{(a,b)\in\mathcal{T}_X} f(a,b) \geq 5\}\end{aligned}$$

In this approach, the burden of deciding the contribution of each selected feature towards modeling the data falls to the parameter estimation algorithm.

3.2.5. Parameter Estimation Each training set $\mathcal{T}_X$ is used to estimate the parameters of a corresponding probability model p_X of the form (2), where $X \in \{$ TAG, CHUNK, BUILD, CHECK $\}$. Each feature f_j corresponds to a parameter α_j, which can be viewed as a "weight" that reflects the importance or usefulness of the feature.

The parameters $\{\alpha_1 \ldots \alpha_k\}$ of each model are found automatically with the *Generalized Iterative Scaling* algorithm (Darroch and Ratcliff, 1972), which is summarized below:

1. Add a "correction" feature f_{k+1} to the model, defined as

$$f_{k+1}(a,b) = C - \sum_{j=1}^{k} f_j(a,b)$$

   where C is some constant ≥ 1 such that for any (a,b) pair:

$$\sum_{j=1}^{k+1} f_j(a,b) = C$$

2. Estimate the parameters using the following iterative algorithm:

$$\alpha_j^{(0)} = 1$$
$$\alpha_j^{(n+1)} = \alpha_j^{(n)} [\frac{E_{\tilde{p}} f_j}{E_{p^{(n)}} f_j}]^{\frac{1}{C}}$$

where

$$E_{p^{(n)}} f_j = \sum_{(a,b)} \tilde{p}(b) p^{(n)}(a|b) f_j(a,b)$$
$$p^{(n)}(a|b) = \frac{1}{Z(b)} \prod_{j=1}^{l} (\alpha_j^{(n)})^{f_j(a,b)}$$

The algorithm guarantees that the likelihood of the training set is non-decreasing, i.e., $L(p^{n+1}) \geq L(p^n)$, and that the sequence $\{p^n | n = 1, 2, \ldots\}$ will eventually converge to p^*, the maximum likelihood estimate for models of form (2).

In practice, the parameter updates can be stopped after some fixed number of iterations (e.g., 100) or when $L(p^{n+1}) - L(p^n) < T$ where T is some small heuristically set threshold. The GIS algorithm is applied separately to the training sets $\mathcal{T}_X$ to create the models p_X, where $X \in \{$ TAG, CHUNK, BUILD, CHECK $\}$.

3.2.6. Scoring Parse Trees We then use the models p_{TAG}, p_{CHUNK}, p_{BUILD}, and p_{CHECK} to define a function score, which the search procedure uses to rank derivations of incomplete and complete parse trees. For notational convenience, define q as follows

$$q(a|b) = \begin{cases} p_{\mathrm{TAG}}(a|b) & \text{if } a \text{ is an action from TAG} \\ p_{\mathrm{CHUNK}}(a|b) & \text{if } a \text{ is an action from CHUNK} \\ p_{\mathrm{BUILD}}(a|b) & \text{if } a \text{ is an action from BUILD} \\ p_{\mathrm{CHECK}}(a|b) & \text{if } a \text{ is an action from CHECK} \end{cases}$$

Let $\mathsf{deriv}(T) = \{a_1, \ldots, a_n\}$ be the derivation of a parse T, where T is not necessarily complete, and where each a_i is an action of some tree-building procedure. By design, the tree-building procedures guarantee that $\{a_1, \ldots, a_n\}$ is the only derivation for the parse T. Then the score of T is merely the product of the conditional probabilities of the individual actions in its derivation:

$$\mathsf{score}(T) = \prod_{a_i \in \mathsf{deriv}(T)} q(a_i | b_i)$$

where b_i is the context in which a_i was decided.

3.3. Search

The search heuristic attempts to find the best parse T^*, defined as:

$$T^* = \operatorname*{argmax}_{T \in \mathsf{trees}(S)} \mathsf{score}(T)$$

```
advance:    d × Q ⟶ d_1 ... d_m     /* Applies relevant tree building
                                     procedure to d and returns list of new
                                     derivations whose action probabilities
                                     pass the threshold Q */
insert:     d × h ⟶ void            /* inserts d in heap h */
extract:    h ⟶ d                   /* removes and returns derivation in h
                                     with highest score */
completed:  d ⟶ {true,false}        /* returns true if and only if d is a
                                     complete derivation */

M = 20
K = 20
Q = .95
C = <empty heap>          /* Heap of completed parses */
h_0 =<input sentence>     /* h_i contains derivations of length i */
while ( |C| < M )
      if ( ∀i, h_i is empty )
         then break
      i = max{i | h_i is non-empty}
      sz = min(K, |h_i|)
      for j = 1 to sz
          d_1 ... d_p = advance( extract(h_i), Q )
          for q = 1 to p
              if (completed(d_q))
                 then insert(d_q, C)
                 else insert(d_q, h_{i+1})
```

Figure 10. Top K BFS Search Heuristic

where $\mathsf{trees}(S)$ are all the complete parses for an input sentence S.

The heuristic employs a breadth-first search (BFS) which does not explore the entire frontier, but rather, explores only at most the top K scoring incomplete parses in the frontier, and terminates when it has found M complete parses, or when all the hypotheses have been exhausted. Furthermore, if $\{a_1 \dots a_n\}$ are the possible actions for a given procedure on a derivation with context b, and they are sorted in decreasing order according to $q(a_i|b)$, we only consider exploring those actions $\{a_1 \dots a_m\}$ that hold most of the probability mass, where m is defined as follows:

$$m = \max_m \sum_{i=1}^{m} q(a_i|b) < Q$$

and where Q is a threshold less than 1. The search also uses a *tag dictionary*, described in Ratnaparkhi (1996), that is constructed from training data and reduces

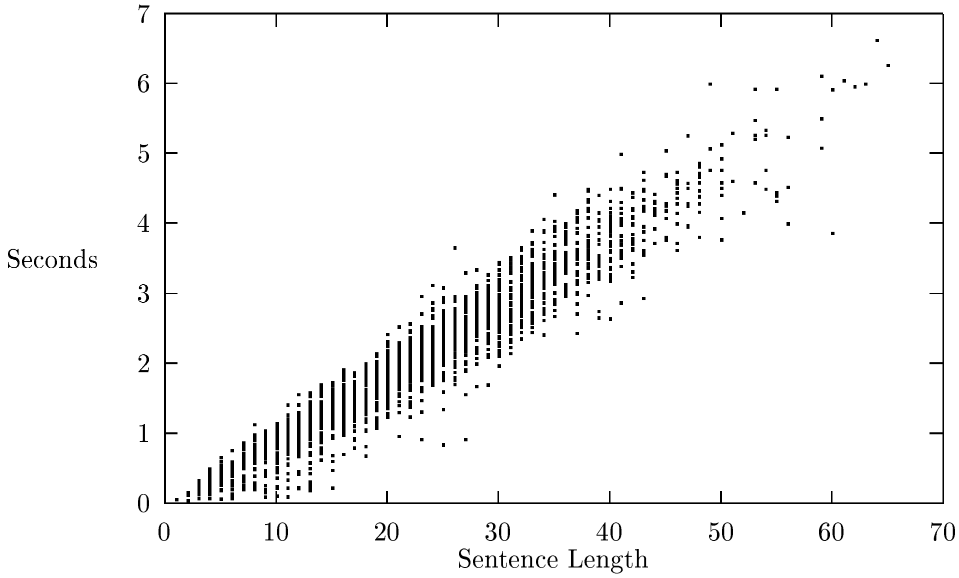


Figure 11. Observed running time of top K BFS on Section 23 of Penn Treebank WSJ, using one 167Mhz UltraSPARC processor and 256MB RAM of a Sun Ultra Enterprise 4000.

the number of actions explored by the tagging model. Thus there are three parameters for the search heuristic, namely K,M, and Q and all experiments reported in this paper use $K = 20$, $M = 20$, and $Q = .95$[1]. Figure 10 describes the top K BFS and the semantics of the supporting functions.

It should be emphasized that if $K > 1$, the parser does not commit to a single POS or chunk assignment for the input sentence before building constituent structure. All three of the passes described in section 3.1 are integrated in the search, i.e., when parsing a test sentence, the input to the second pass consists of K of the top scoring distinct POS tag assignments for the input sentence. Likewise, the input to the third pass consists of K of the top scoring distinct chunk and POS tag assignments for the input sentence.

The top K BFS described above exploits the observed property that the individual steps of correct derivations tend to have high probabilities, and thus avoids searching a large fraction of the search space. Since, in practice, it only does a constant amount of work to advance each step in a derivation, and since derivation lengths are roughly proportional to the sentence length, we would expect it to run in linear observed time with respect to sentence length. Figure 11 confirms our assumptions about the linear observed running time.

Table 4. Sizes of Training Events, Actions, and Features

Procedure	Number of Training Events	Number of Actions	Number of Features
TAG	935655	43	119910
CHUNK	935655	41	230473
CHECK	1097584	2	182474
BUILD	1097584	52	532814

4. Experiments

Experiments were conducted on a treebank that is widely used in the statistical natural language processing community, namely, the Wall St. Journal treebank (release 2) from the University of Pennsylvania (Marcus et al., 1994). The maximum entropy parser was trained on sections 2 through 21 (roughly 40000 sentences) of the Wall St. Journal corpus, and tested on section 23 (2416 sentences) for comparison with other work. Table 4 describes the number of training events extracted from the Wall St. Journal corpus, the number of actions in the resulting probability models, and the number of selected features in the resulting probability models. It took roughly 30 hours to train all the probability models, using one 167 Mhz Sun UltraSPARC processor and 1 Gb of disk space. Only the words, part-of-speech tags, constituent labels, and constituent boundaries of the Penn treebank were used for training and testing. The other annotation, such as the *function tags* that indiciate semantic properties of constituents, and the *null elements* that indicate traces and coreference, were removed for both training and testing. Previous literature on statistical parsing has used the following measures, based on those proposed in Black et al. (1991), for comparing a proposed parse P with the corresponding correct treebank parse T:

$$\text{Recall} = \frac{\#\text{ correct constituents in } P}{\#\text{ constituents in } T}$$

$$\text{Precision} = \frac{\#\text{ correct constituents in } P}{\#\text{ constituents in } P}$$

A constituent in P is "correct" if there exists a constituent in T of the same label that spans the same words.[2] Table 5 shows results using these measures, as well as results using the slightly more forgiving measures used in Magerman (1995). Table 5 shows that the maximum entropy parser compares favorably to other state-of-the-art systems (Magerman, 1995; Collins, 1996; Goodman, 1997; Charniak, 1997; Collins, 1997) and shows that only the results of Collins (1997) are better in both precision and recall. The parser of Hermjakob and Mooney (1997) also performs well (90% labelled precision and recall) on the Wall St. Journal domain, but uses a test set comprised of sentences with only frequent words and recovers a different

Table 5. Results on 2416 sentences of section 23 (0 to 100 words in length) of the WSJ Treebank. Evaluations marked with $^\diamond$ ignore quotation marks. Evaluations marked with * collapse the distinction between the constituent labels `ADVP` and `PRT`, and ignore *all* punctuation.

Parser	Precision	Recall
Maximum Entropy$^\diamond$	86.8%	85.6%
Maximum Entropy*	87.5%	86.3%
(Magerman, 1995)*	84.3%	84.0%
(Collins, 1996)*	85.7%	85.3%
(Goodman, 1997)*	84.8%	85.3%
(Charniak, 1997)*	86.7%	86.6%
(Collins, 1997)*	88.1%	87.5%

Table 6. Speed and accuracy on 5% random sample of test set, as a function of search parameters K and M

K, M	Seconds/Sentence	Precision	Recall
20	2.07	87.9	87.1
15	1.58	87.7	86.9
10	1.07	87.7	86.9
7	0.76	87.4	86.6
5	0.56	87.3	86.8
3	0.35	86.1	86.1
1	0.14	82.4	83.4

form of annotation, and is therefore not comparable to the parsers in Table 5. Figure 12 shows the effects of training data size versus performance, and Table 6 shows the effect of varying the search parameters K and M on the parser's speed and accuracy. Parsing accuracy degrades as K and M are reduced, but even with $K = 1$ and $M = 1$, accuracy is over 82% precision and recall.

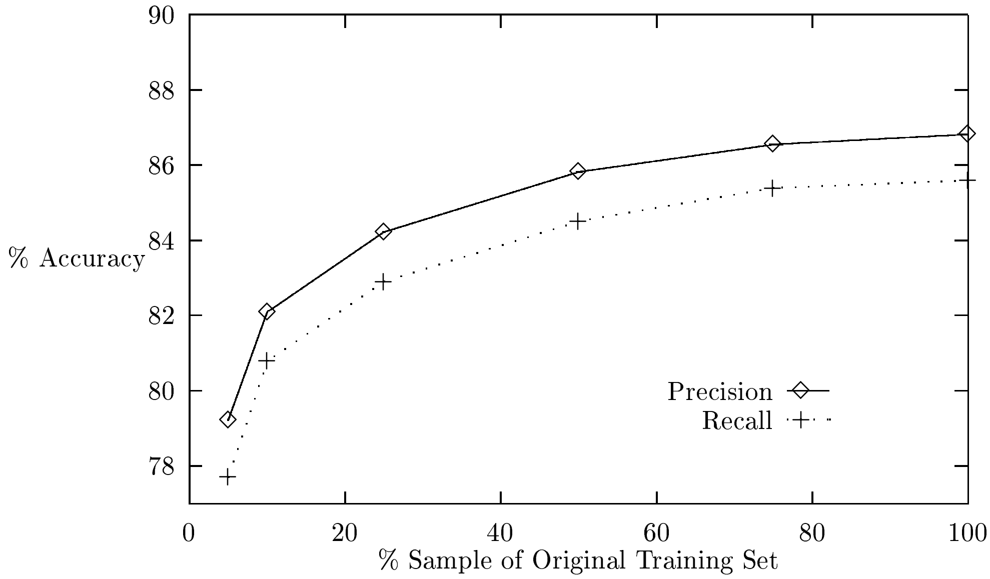


Figure 12. Performance on section 23 as a function of training data size. The X axis represents random samples of different sizes from sections 2 through 21 of the Wall St. Journal corpus.

4.1. Portability

Portability across domains is an important concern, since corpus-based methods will suffer in accuracy if they are tested in a domain that is unrelated to the one in which they are trained (e.g., see Sekine (1997)). Since treebank construction is a time-consuming and expensive process, it is unlikely (in the near future) that treebanks will exist for every domain that we could conceivably want to parse. It then becomes important to quantify the potential loss in accuracy when training on a treebanked domain, like the Wall St. Journal, and testing on a new domain. The experiments in this section address the following two practical questions :

- How much accuracy is lost when the parser is trained on the Wall St. Journal domain, and tested on another domain (compared to when the parser is trained and tested on the Wall St. Journal) ?
- How much does a small amount of additional training material (2000 sentences) on a new domain help the parser's accuracy on the new domain ?

The new domains, namely "Magazine & Journal Articles", "General Fiction", and "Adventure Fiction", are from the Brown corpus (Francis and Kucera, 1982), a collection of English text from Brown University that represents a wide variety of

Table 7. Description of training and test sets

Name	Description	Category
WSJ.train	Sections 2 through 21 of the WSJ corpus	Financial News
G.train	First 2000 sentences of section G in Brown corpus	Magazine Articles
G.test	Remaining 1209 sentences of section G in Brown corpus	Magazine Articles
K.train	First 2000 sentences of section K in Brown corpus	General Fiction
K.test	Remaining 2006 sentences of section K in Brown corpus	General Fiction
N.train	First 2000 sentences of section N in Brown corpus	Adventure Fiction
N.test	Remaining 2121 sentences of section N in Brown corpus	Adventure Fiction

Table 8. Portability Experiments on the Brown corpus. See Table 7 for the training and test sets.

Strategy #	Description	Test Corpus Accuracy (Precision/Recall) G	K	N	Avg. Accuracy (Precision/Recall)
Strategy 1	Train on WSJ.train, test on X.test	80.2%/79.5%	79.1%/78.8%	80.6%/79.9%	80.0%/79.4%
Strategy 2	Train on WSJ.train + X.train, test on X.test	81.0%/80.5%	80.9%/80.3%	82.0%/81.0%	81.3%/80.6%
Strategy 3	Train on X.train, test on X.test	78.2%/76.3%	77.7%/76.7%	78.7%/77.6%	78.2%/76.9%

different domains. These domains have been annotated in a convention similar to the text of the Wall St. Journal treebank, as part of the Penn treebank project.

Table 8 describes the results of several different training schemes, and Table 7 describes the training and test corpora. The feature sets of the parser were not changed in any way when training from the Brown corpus domains. According to Table 8, the training schemes for parsing a new domain D, ranked in order from best to worst, are:

1. Strategy 2: Train on a mixture of a lot of Wall St. Journal (WSJ) and a little of D
2. Strategy 1: Train on a lot of WSJ
3. Strategy 3: Train on a little of D

All experiments on a particular new domain (G, K, and N) are controlled to use the same test set, and the additional training sets G.train, K.train, and N.train all consist of 2000 sentences from their respective domain. Compared to the accuracy achieved when training and testing on the Wall St. Journal (86.8% precision/85.6% recall as shown in Table 5), we conclude that:

- on average, we lose 6.8% precision and 6.2% recall when training on the Wall St. Journal and testing on the Brown corpus (strategy 1),

- on average, we lose 5.5 % precision and 5% recall when training on the Wall St. Journal and the domain of interest, and testing on that same domain (strategy 2).

The discussion thus far has omitted one other possibility, namely, that the lower Brown corpus performance in strategies 1 and 2 is due to some inherent difficulty in parsing the Brown corpus text, and not to the mismatch in training and test data. A quick glance at Figure 12 and Table 8 dispels this possibility, since training on roughly 2000 sentences of the Wall St. Journal yields 79% precision and 78% recall, which is only slightly higher (1%) than the results on the Brown corpus under identical circumstances (strategy 3), roughly 78% precision 77% recall. It follows that the Brown corpus is only slightly more difficult to parse than the Wall St. Journal corpus, and that the training domain/test domain mismatch must account for most of the accuracy loss when using strategies 1 and 2.

4.2. Reranking the Top N

It is often advantageous to produce the top N parses instead of just the top 1, since additional information can be used in a secondary model that re-orders the top N and hopefully improves the quality of the top ranked parse. Suppose there exists a *perfect* reranking scheme that, for each sentence, magically picks the *best* parse from the top N parses produced by the maximum entropy parser, where the *best* parse has the highest average precision and recall when compared to the treebank parse. The performance of this perfect scheme is then an upper bound on the performance of an actual reranking scheme that might be used to reorder the top N parses. Figure 13 shows that the perfect scheme would achieve roughly 93% precision and recall, which is a dramatic increase over the top 1 accuracy of 87% precision and 86% recall. Figure 14 shows that the "Exact Match", which counts the percentage of times the proposed parse P is identical (excluding POS tags) to the treebank parse T, rises substantially to about 53% from 30% when the perfect scheme is applied. It is not surprising that the accuracy improves by looking at the top N parses, but it is suprising—given the thousands of partial derivations that are explored and discarded—that the accuracy improves drastically by looking at *only the top 20 completed parses.* For this reason, research into reranking schemes appears to be a promising and practical step towards the goal of improving parsing accuracy.

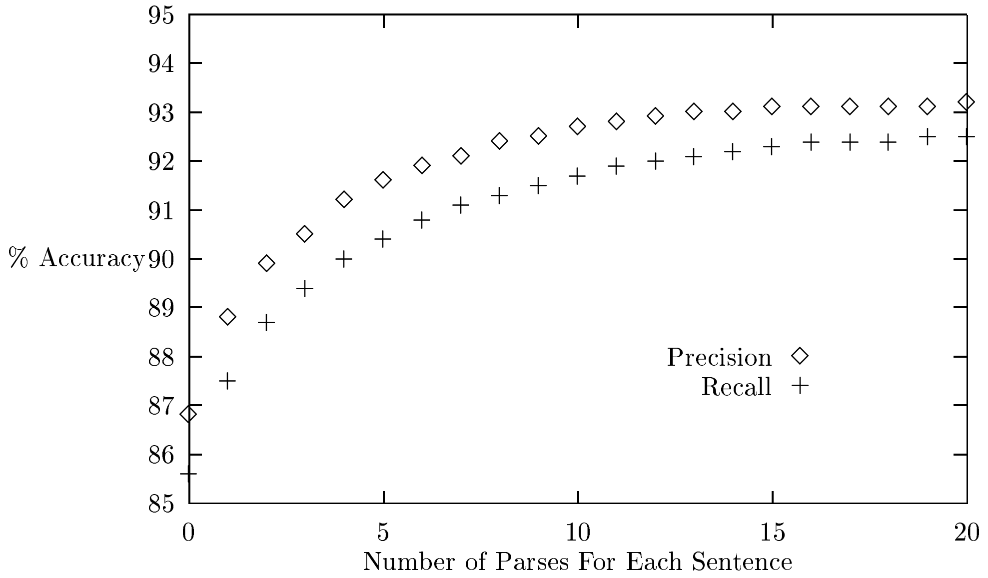


Figure 13. Precision & recall of a "perfect" reranking scheme for the top N parses of section 23 of the WSJ Treebank, as a function of N. Evaluation ignores quotation marks.

5. Comparison With Previous Work

When compared to other parsers, the accuracy of the maximum entropy parser is state-of-the-art. It performs slightly better than or equal to most of the other systems compared in Table 5, and performs only slightly worse than Collins (1997). However, the differences in accuracy are fairly small, and it is unclear if the differences will matter to the performance of applications that require parsed input. The main advantage of the maximum entropy parser is not its accuracy, but that it achieves the accuracy using only simple facts about data that have been derived from linguistically obvious intuitions about parsing. As a result, the evidence it needs can be specified concisely, and the method can be re-used from other tasks, resulting in a minimum amount of effort on the part of the experimenter.

Furthermore, the maximum entropy parser combines some of the best aspects of other work. For example, the parsers of Black et al. (1993), Jelinek et al. (1994), Magerman (1995) use a general learning technique—decision trees—to learn parsing actions, and need to represent words as bitstrings derived from a statistical word clustering technique. The maximum entropy parser also uses a general learning technique, but uses natural linguistic representations of words and constituents, and therefore does not require a (typically expensive) word clustering procedure.

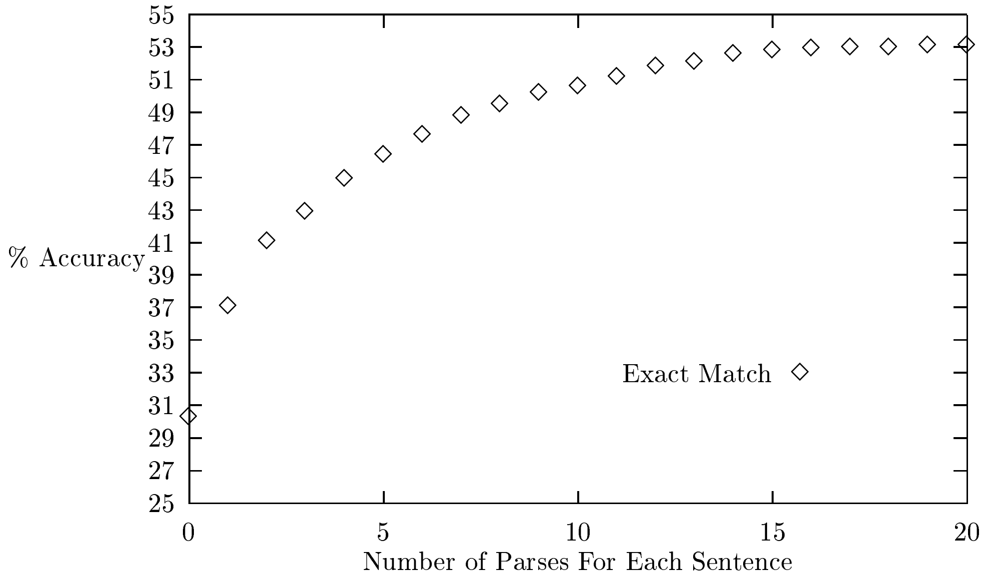


Figure 14. Exact match of a "perfect" reranking scheme for the top N parses of section 23 of the WSJ Treebank, as a function of N. Evaluation ignores quotation marks.

Other parsers, like those of Collins (1996), Goodman (1997), Charniak (1997), Collins (1997) use natural linguistic representations of words and constituents, but do not use general machine learning techniques. Instead, they use custom-built statistical models that combine evidence in clever ways to achieve high parsing accuracies. While it is always possible to tune such methods to maximize accuracy, the methods are specific to the parsing problem and require non-trivial research effort to develop. In contrast, the maximum entropy parser uses an existing modeling framework that is essentially independent of the parsing task, and saves the experimenter from designing a new, parsing-specific statistical model.

In general, more supervision typically leads to higher accuracy. For example, Collins (1997) uses the semantic tags in the Penn treebank while the other, slightly less accurate parsers in Table 5 discard this information. Also, Hermjakob and Mooney (1997) uses a hand-constructed knowledge base and subcategorization table and report 90% labelled precision and recall, using a different test set and evaluation method. The additional information used in these approaches, as well as the word clusters used in Magerman (1995), could in theory be implemented as features in the maximum entropy parser. Further research is needed to see if such additions to the parser's representation will improve the parser's accuracy.

The portability of all the parsers discussed here is limited by the availability of treebanks. Currently, few treebanks exist, and constructing a new treebank requires a tremendous amount of effort. It is likely that all current corpus-based parsers will parse text less accurately if the domain of the text is not similar to the domain of the treebank that was used to train the parser.

6. Conclusion

The maximum entropy parser achieves state-of-the-art parsing accuracy, and minimizes the human effort necessary for its construction through its use of both a general learning technique, and a simple representation derived from a few intuitions about parsing. Those results which exceed those of the parser presented here require much more human effort in the form of additional resources or annotation. In practice, it parses a test sentence in linear time with respect to the sentence length. It can be trained from other domains without modification to the learning technique or the representation. Lastly, this paper clearly demonstrates that schemes for reranking the top 20 parses deserve research effort since they could yield vastly better accuracy results.

The high accuracy of the maximum entropy parser also has interesting implications for future applications of general machine learning techniques to parsing. It shows that the procedures and actions with which a parser builds trees can be designed independently of the learning technique, and that the learning technique can utilize the exactly same sorts of information, e.g., words, tags, and constituent labels, that might normally be used in a more traditional, non-statistical natural language parser. This implies that it is feasible to use maximum entropy models and other general learning techniques to drive the actions of other kinds of parsers trained from more linguistically sophisticated treebanks. Perhaps a better combination of learning technique, parser, and treebank will exceed the current state-of-the-art parsing accuracies.

7. Acknowledgments

The author would like to thank Mike Collins and Mitch Marcus of the University of Pennsylvania for their many helpful comments on this work. The author would also like to thank the three anonymous reviewers of this paper for their constructive comments. This work was supported by ARPA grant N66001-94C-6043.

Notes

1. The parameters K,M, and Q were optimized for speed and accuracy on a "development set" which is separate from the training and test sets.
2. The precision and recall measures do not count part-of-speech tags as constituents.

References

Abney, S. (1991). Parsing By Chunks. In Berwick, R., Abney, S., and Tenny, C. (Eds.), *Principle-Based Parsing*. Kluwer Academic Publishers.

Aho, A. V., Sethi, R., and Ullman, J. D. (1988). *Compilers : Principles Techniques and Tools*. Addison Wesley.

Allen, J. (1995). *Natural Language Understanding*. Benjamin/Cummings Publishing.

Berger, A., Della Pietra, S. A., and Della Pietra, V. J. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, *22*(1), 39–71.

Black, E., et al. (1991). A Procedure for Quantitatively Comparing the Syntactic Coverage of English Grammars. In *Proceedings of the February 1991 DARPA Speech and Natural Language Workshop*, pp. 306–311.

Black, E., Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., and Roukos, S. (1993). Towards History-based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the 31st Annual Meeting of the ACL* Columbus, Ohio.

Briscoe, T., and Carroll, J. (1993). Generalized Probabilistic LR Parsing of Natural Language (Corpora) with Unification-Based Grammars. *Computational Linguistics*, *19*(1).

Charniak, E. (1997). Statistical Parsing with a Context-free Grammar and Word Statistics. In *Fourteenth National Conference on Artificial Intelligence* Providence, Rhode Island.

Church, K. (1988). A stochastic parts program and noun phrase chunker for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing*.

Collins, M. (1997). Three Generative, Lexicalised Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the ACL, and 8th Conference of the EACL* Madrid, Spain. ACL.

Collins, M. J. (1996). A New Statistical Parser Based on Bigram Lexical Dependencies. In *Proceedings of the 34th Annual Meeting of the ACL*.

Darroch, J. N., and Ratcliff, D. (1972). Generalized Iterative Scaling for Log-Linear Models. *The Annals of Mathematical Statistics*, *43*(5), 1470–1480.

Della Pietra, S., Della Pietra, V., and Lafferty, J. (1997). Inducing Features of Random Fields. *IEEE Transactions Pattern Analysis and Machine Intelligence*, *19*(4).

Francis, W. N., and Kucera, H. (1982). *Frequency analysis of English usage: lexicon and grammar*. Houghton Mifflin, Boston.

Goodman, J. (1997). Probabilistic Feature Grammars. In *Proceedings of the International Workshop on Parsing Technologies*.

Hermjakob, U., and Mooney, R. J. (1997). Learning Parse and Translation Decision From Examples With Rich Context. In *Proceedings of the 35th Annual Meeting of the ACL, and 8th Conference of the EACL* Madrid, Spain. ACL.

Jaynes, E. T. (1957). Information Theory and Statistical Mechanics. *Physical Review*, *106*, 620–630.

Jelinek, F., Lafferty, J., Magerman, D. M., Mercer, R., Ratnaparkhi, A., and Roukos, S. (1994). Decision Tree Parsing using a Hidden Derivational Model. In *Proceedings of the Human Language Technology Workshop*, pp. 272–277 Plainsboro, N.J. ARPA.

Lau, R., Rosenfeld, R., and Roukos, S. (1993). Adaptive Language Modeling Using The Maximum Entropy Principle. In *Proceedings of the Human Language Technology Workshop*, pp. 108–113. ARPA.

Magerman, D. M. (1995). Statistical Decision-Tree Models for Parsing. In *Proceedings of the 33rd Annual Meeting of the ACL*.

Marcus, M. P. (1980). *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge, Mass.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1994). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, *19*(2), 313–330.

Ramshaw, L. A., and Marcus, M. P. (1995). Text Chunking using Transformation-Based Learning. In Yarowsky, D., and Church, K. (Eds.), *Proceedings of the Third Workshop on Very Large Corpora* Cambridge, Massachusetts.

Ratnaparkhi, A. (1996). A Maximum Entropy Part of Speech Tagger. In Brill, E., and Church, K. (Eds.), *Conference on Empirical Methods in Natural Language Processing* University of Pennsylvania.

Rosenfeld, R. (1996). A Maximum Entropy Approach to Adaptive Statistical Language Modeling. *Computer, Speech, and Language*, *10*.

Sekine, S. (1997). The Domain Dependence of Parsing. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp. 96–102 Washington D.C.

Weischedel, R., Meteer, M., Schwartz, R., Ramshaw, L., and Palmucci, J. (1993). Coping With Ambiguity and Unknown Words through Probabilistic Models. *Computational Linguistics*, *19*(2), 359–382.