Discriminative Reranking for Natural Language Parsing

Michael Collins MCOLLINS@RESEARCH.ATT.COM
AT&T Labs–Research, Rm A-253, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932

Abstract

This paper considers approaches which rerank the output of an existing probabilistic parser. The base parser produces a set of candidate parses for each input sentence, with associated probabilities that define an initial ranking of these parses. A second model then attempts to improve upon this initial ranking, using additional features of the tree as evidence. We describe and compare two approaches to the problem: one based on Markov Random Fields, the other based on boosting approaches to reranking problems. The methods were applied to reranking output of the parser of Collins (1999) on the Wall Street Journal corpus, with a 13% relative decrease in error rate.

1. Introduction

Machine-learning approaches to natural language parsing have recently shown some success in complex domains such as newswire text. Many of these methods fall into the general category of history-based models, where a parse tree is represented as a derivation (sequence of decisions) and the probability of the tree is then calculated as a product of decision probabilities. While these approaches have many advantages, it can be awkward to encode some constraints within this framework. It is often easy to think of features which might be useful in discriminating between candidate trees for a sentence, but much more difficult to alter the derivation to take these features into account.

This paper considers approaches which rerank the output of an existing probabilistic parser. The base parser produces a set of candidate parses for each input sentence, with associated probabilities that define an initial ranking of these parses. A second model then attempts to improve upon this initial ranking, using additional features of the tree as evidence. The strength of our approach is that it allows a tree to be represented as an arbitrary set of features, without concerns about how these features interact or overlap, and without the need to define a derivation which takes these features into account.

The problems with history-based models, and the desire to be able to specify features as arbitrary predicates of the entire tree, have been noted before. In particular, previous work (Abney 1997; Della Pietra, Della Pietra & Lafferty 1997; Johnson et al. 1999) has investigated the use of Markov Random Fields (MRFs), or log-linear models, as probabilistic models for parsing and other NLP tasks. The first method we discuss is based on a feature selection method within the MRF framework. The second approach is based on the application of boosting models for ranking problems (Freund et al. 1998). Previous work (Friedman, Hastie & Tibshirani 1998) has drawn connections between log-linear models and boosting for *classification* problems; one contribution of this paper is to draw similar connections between the two approaches to ranking problems. Efficiency is an important issue in our problem – the training data consists of around 1 million trees,[1] and there are approximately 500,000 distinct features. We show that a naive implementation of the boosting method is computationally expensive, but that the sparse nature of the feature space means that a much improved algorithm can be derived. The MRF algorithm also benefits from the sparseness of the feature space, but for reasons we will discuss is still much less efficient than the boosting approach.

We applied our method to parsing the Wall Street Journal treebank (Marcus, Santorini & Marcinkiewicz 1993). The baseline model, that of Collins (1999), achieved 88.1/88.3% recall/precision on this task. The new model achieves 89.6/89.9% recall/precision, a 13% relative decrease in error. Although this paper concentrates on parsing, many other problems in natural language processing or speech recognition can also be framed as reranking problems, so the results of this paper should be quite broadly applicable.

1.1 History-Based Models

Before discussing the reranking approaches, we will describe *history-based* models (Black et al. 1992). They are important for a few reasons. First, at present the best performing parsers on the WSJ treebank (Ratnaparkhi 1997; Charniak 1997, 1999; Collins 1997, 1999) are all cases of history-based models. Many systems applied to part-of-speech tagging, speech recognition and other language or speech tasks also fall into this class of model. Second, a particular history-based model (that of Collins (1999)) will be used as the initial model for our approach. Finally, it is important to describe history-based models – and to understand their limitations – to motivate our departure from them.

Parsing can be framed as a supervised learning task, to induce a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ given training examples

[1]Training data includes 36,000 sentences, with an average of over 27 trees per sentence.

$\langle X_i, Y_i \rangle$ where $X_i \in \mathcal{X},\ Y_i \in \mathcal{Y}$. We define $\mathcal{Y}(X) \subset \mathcal{Y}$ to be the set of candidates for a given input X. In the parsing problem X is a sentence, and $\mathcal{Y}(X)$ is a set of candidate trees for that sentence. A particular characteristic of the problem is the complexity of $\mathcal{Y}(X)$: $\mathcal{Y}(X)$ can be very large, and each member of $\mathcal{Y}(X)$ has rich internal structure. This contrasts with "typical" classification problems where $\mathcal{Y}(X)$ is a fixed, small set, for example $\{-1, +1\}$ in binary classification problems.

In probabilistic approaches, a model is defined which assigns a probability $P(Y, X)$ to each (Y, X) pair. The most likely parse for each sentence X is then $\arg\max_{Y \in \mathcal{Y}(X)} P(Y, X)$. This leaves the question of how to define $P(Y, X)$. In history-based approaches, a one-to-one mapping is defined between each pair (Y, X) and a decision sequence $\langle d_1 \ldots d_n \rangle$. The sequence $\langle d_1 \ldots d_n \rangle$ can be thought of as the sequence of moves that build (Y, X) in some canonical order. Given this mapping, the probability of a tree can be written as

$$P(Y, X) \quad = \quad \prod_{i=1 \ldots n} P\left(d_i | \Phi\left(d_1 \ldots d_{i-1}\right)\right)$$

$\left(d_1 \ldots d_{i-1}\right)$ is the *history* for the i'th decision. Φ is a function which groups histories into equivalence classes, thereby making independence assumptions in the model.

Probabilistic Context-Free Grammars (PCFGs) are one example of a history-based model. The decision sequence $\langle d_1 \ldots d_n \rangle$ is defined as the sequence of rule expansions in a top-down, left-most derivation of the tree. The history is equivalent to a partially built tree, and Φ picks out the left-most non-terminal in the fringe of this tree, making the assumption that $P(d_i | d_1 \ldots d_{i-1})$ depends only on the non-terminal being expanded. In the resulting model a tree with rules $\langle \gamma_i \rightarrow \beta_i \rangle$ is assigned a probability $\prod_{i=1}^{n} P(\beta_i | \gamma_i)$.

Our base model, that of Collins (1999), is also a history based model. It can be considered to be a type of PCFG, where the rules are lexicalized. An example rule would be:

```
VP(saw) -> VBD(saw) NP-C(her) NP(today)
```

Lexicalization leads to a very large number of rules; to make the number of parameters manageable the generation of the right hand side of a rule is broken down into a number of decisions. First the head non-terminal (`VBD` in the above example) is chosen. Next, left and right subcategorization frames are chosen ({} and {NP-C}). Non-terminal sequences to the left and right of the `VBD` are chosen (an empty sequence to the left, ⟨`NP-C,NP`⟩ to the right). Finally, the lexical heads of the modifiers are chosen (*her* and *today*). Each of these decisions has an associated probability conditioned on the left hand side of the rule (`VP(saw)`) and other information in some cases.

History-based approaches lead to models where the log probability of a parse-tree can be written as a linear sum of parameters α_s multiplied by features h_s. Each feature $h_s(X, Y)$ is the count of a different "event", or fragment within the tree. As an example, consider a PCFG with rules $\langle \gamma_s \rightarrow \beta_s \rangle$ for $1 \leq s \leq m$. If $h_s(X, Y)$ is the number of times $\langle \gamma_s \rightarrow \beta_s \rangle$ is seen in the tree, and $\alpha_s = \log P(\beta_s | \gamma_s)$ is the parameter associated with that rule, then

$$\log P(Y, X) = \sum_{s=1}^{m} \alpha_s h_s(X, Y)$$

All models considered in this paper take this form, although in the boosting models the score for a parse is not a log probability. The features h_s define an m-dimensional vector of counts which represent the tree. The parameters α_s represent the influence of each feature on the score of a tree.

A drawback of history-based models is that the choice of derivation has a profound influence on the parameters of the model. (Similar observations have been made in the related cases of belief networks (Pearl 1988), and language models for speech recognition (Rosenfeld 1997).) When designing a model, it is often easy to think of features which might be useful in ranking trees in order of plausibility, but much more difficult to modify the derivation so that these features are included in the model. In an ideal situation we would be able to encode arbitrary features h_s, thereby keeping track of counts of arbitrary fragments within parse trees, without having to worry about formulating a derivation that included these features.

To take a concrete example, consider part-of-speech tagging using a Hidden Markov Model. We might have the intuition that almost every sentence has at least one verb, and therefore that sequences including at least one verb should have increased scores under the model. Encoding this constraint in a compact way in an HMM takes some ingenuity.[2] In contrast, it would be trivial to implement a feature $h_s(X, Y)$ which is 1 if Y contains a verb, 0 otherwise.

2. Reranking Approaches

2.1 Problem Definition

We use the following notation in the rest of this paper:

- $x_{i,j}$ is the j'th parse of the i'th sentence. There are n sentences, and n_i parses for the i'th sentence, i.e., $1 \leq i \leq n$ and $1 \leq j \leq n_i$. Each $x_{i,j}$ contains both the tree and the underlying sentence (i.e. each $x_{i,j}$ is a member of $\mathcal{X} \times \mathcal{Y}$ by the notation of section 1.1).
- $Score(x_{i,j})$ is the "score" for parse $x_{i,j}$, a measure of the similarity of $x_{i,j}$ to the gold-standard parse.
- Without loss of generality, we assume x_{i1} to be the highest scoring parse for the i'th sentence.[3] More precisely, for all $i, 2 \leq j \leq n_i$, $Score(x_{i,1}) > Score(x_{i,j})$. (Note that $x_{i,1}$ may not be identical to

[2]The obvious approach, to add to each state the information about whether or not a verb has been generated in the history, doubles the number of states (and parameters) in the model

[3]In the event that multiple parses get the same, highest score the parse with the highest value of log-likelihood L under the baseline model is taken as $x_{i,1}$.

the gold-standard parse — in some cases the parser may fail to propose the correct parse anywhere in its list of candidates.)

- $Q(x_{i,j})$ is the probability that the base parsing model assigns to parse $x_{i,j}$. $L(x_{i,j}) = \log Q(x_{i,j})$ is the log probability.
- There is a separate test set of parses, $y_{i,j}$, with similar definitions of $Score$, Q and L.

The task is to learn a ranking function, $F(x_{i,j})$. The base ranking function, which we would like to improve upon, is $F(x_{i,j}) = L(x_{i,j})$. The performance of a given ranking function will be evaluated on test data: if z_i is the index of the top-ranked parse under F on the i'th test example

$$z_i = \arg\max_{j=1\ldots n_i} F(y_{i,j})$$

then the "score" of F is $\sum_i Score(y_{i,z_i})$ and the maximum possible score is $\sum_i Score(y_{i,1})$.

We will consider a particular form for F in this paper. We assume a vector of $m+1$ parameters, $\bar{\alpha} = \{\alpha_0, \alpha_1 \ldots \alpha_m\}$. The ranking function is defined as

$$F(x_{i,j}, \bar{\alpha}) = \alpha_0 L(x_{i,j}) + \sum_{s=1}^{m} \alpha_s h_s(x_{i,j})$$

Each h_s is an indicator function, for example

$$h_s(x) = \begin{cases} 1 & \text{if } x \text{ contains the rule } \langle S \rightarrow NP \;\; VP \rangle \\ 0 & \text{otherwise} \end{cases} .$$

The restriction to binary valued features is important for the simplicity and efficiency of the algorithms.[4]

The learning task is to find parameter settings for $\bar{\alpha}$ which lead to good scores on test data; the parameters will be set using training data examples as evidence. We now discuss how to set these parameters. First we discuss loss functions $Loss(\bar{\alpha})$ which can be used to drive the training process. We then go on to describe optimization methods for the different loss functions.

2.2 Loss Functions

The loss functions we consider are all related to the number of ranking errors a function F makes on the training set. The ranking error rate is the number of times a lower scoring parse is (incorrectly) ranked above the best parse:

$$\begin{aligned} Error(\bar{\alpha}) &= \sum_i \sum_{j=2}^{n_i} [\![F(x_{i,1}, \bar{\alpha}) < F(x_{i,j}, \bar{\alpha})]\!] \\ &= \sum_i \sum_{j=2}^{n_i} [\![F(x_{i,1}, \bar{\alpha}) - F(x_{i,j}, \bar{\alpha}) < 0]\!] \end{aligned}$$

$[\![x]\!]$ is 1 if x is true, 0 otherwise. Throughout this paper we will refer to the *Margin* on example $x_{i,j}$ as

$$M_{ij}(\bar{\alpha}) = F(x_{i,1}, \bar{\alpha}) - F(x_{i,j}, \bar{\alpha})$$

[4]Note that features tracking the *counts* of different rules can be simulated through several features which take value 1 if a rule is seen $\geq$ 1 time, $\geq$ 2 times, $\geq$ 3 times and so on.

The ranking error is 0 if all margins are positive. The loss functions we discuss all turn out to be direct functions of the margins on training examples.

2.2.1 Log-Likelihood

The first loss function is that suggested by Markov Random Fields, which have been applied to NLP problems by Abney (1997), Della Pietra et al. (1997) and Johnson et al. (1999). As suggested by Johnson et al. (1999), the conditional probability of $x_{i,q}$ being the correct parse for the i'th sentence is defined as

$$P(x_{i,q}) = \frac{e^{F(x_{i,q})}}{\sum_{j=1}^{n_i} e^{F(x_{i,j})}}$$

The log-likelihood of the training data is then

$$\sum_i \log P(x_{i,1}) = \sum_i \log \frac{e^{F(x_{i,1})}}{\sum_{j=1}^{n_i} e^{F(x_{i,j})}}$$

Under maximum likelihood estimation, the parameters $\bar{\alpha}$ would be set to maximize the log-likelihood. Equivalently, we will talk about *minimizing* the *negative* log-likelihood $\sum_i - \log P(x_{i,1})$ (this is for consistency between the log-loss and boosting methods, which now both have an objective function that must be minimized). Some manipulation shows that the negative log-loss is a function of the margins on training data:

$$\begin{aligned} LogLoss(\bar{\alpha}) &= \sum_i - \log \frac{e^{F(x_{i,1})}}{\sum_{j=1}^{n_i} e^{F(x_{i,j})}} \\ &= \sum_i - \log \frac{1}{\sum_{j=1}^{n_i} e^{-(F(x_{i,1}) - F(x_{i,j}))}} \\ &= \sum_i \log \left(1 + \sum_{j=2}^{n_i} e^{-(F(x_{i,1}) - F(x_{i,j}))} \right) \\ &= \sum_i \log \left(1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})} \right) \qquad (1) \end{aligned}$$

2.2.2 Boosting Loss

The next loss function is based on the boosting method described by Schapire and Singer (1998). It is a special case of the general ranking methods described by Freund et al. (1998), with the ranking "feedback" being a simple binary distinction between the highest scoring parse and the other parses. Again, the loss function is a function of the margins on training data:

$$\begin{aligned} BoostLoss(\bar{\alpha}) &= \sum_i \sum_{j=2}^{n_i} e^{-(F(x_{i,1}) - F(x_{i,j}))} \\ &= \sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})} \end{aligned}$$

It can be shown that $BoostLoss(\bar{\alpha}) \geq Error(\bar{\alpha})$, so that minimizing $BoostLoss(\bar{\alpha})$ is closely related to minimizing the number of ranking errors. This follows from the fact that for any x, $e^{-x} \geq [\![x < 0]\!]$, and therefore that

$$\sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})} \geq \sum_i \sum_{j=2}^{n_i} [\![M_{i,j} < 0]\!]$$

2.3 Optimization Methods

The simplest goal would be to find parameter settings $\bar{\alpha}^*$ which minimize a given loss function:

$$\bar{\alpha}^* = \arg\min_{\alpha} Loss(\bar{\alpha})$$

Unfortunately, this method is likely to lead to overtraining – particularly in the domain we are considering, where there are a very large number of features.

Instead, we will focus on feature-selection methods, where the goal is to find a small subset of the features that contribute most to reducing the loss function. The methods are greedy, at each iteration picking the feature h_s with additive weight δ which has the most impact on the loss function. In general, a separate set of instances will be used in cross-validation to choose the stopping point, i.e., to decide on the number of features in the model.

At this point we introduce some notation concerning feature selection methods. We define Upd$(\bar{\alpha}, k, \delta)$ to be an updated parameter vector, with the same parameter values as $\bar{\alpha}$ with the exception of α_k, which is incremented by δ

$$\text{Upd}(\bar{\alpha}, k, \delta) = \{\alpha_0, \alpha_1, \ldots \alpha_k + \delta \ldots, \alpha_m\}$$

The loss for the updated model is $Loss\left(\text{Upd}(\bar{\alpha}, k, \delta)\right)$. Assuming we greedily pick a single feature with some weight to update the model, and given that the current parameter settings are $\bar{\alpha}$, the optimal feature/weight pair (k^*, δ^*) is

$$(k^*, \delta^*) = \arg\min_{k,\delta} Loss\left(\text{Upd}(\bar{\alpha}, k, \delta)\right)$$

The feature selection algorithms we consider take the following form ($\bar{\alpha}^t$ is the parameter vector at the t'th iteration):

1 Initialize $\bar{\alpha}^0$ to some value. (This will generally involve values of zero for $\alpha_1 \ldots \alpha_m$, and a non-zero value for α_0, for example $\bar{\alpha}^0 = \{1, 0, 0, \ldots\}$.)

2 for t = 1 to N (The number of iterations N will be chosen by cross validation):

– Find $(k^*, \delta^*) = \arg\min_{k,\delta} Loss\left(\text{Upd}(\bar{\alpha}^{t-1}, k, \delta)\right)$

– Set $\bar{\alpha}^t = \text{Upd}(\bar{\alpha}^{t-1}, k^*, \delta^*)$

The main computation for both loss functions involves search for the optimal feature/weight pair (k^*, δ^*). Both approaches involve a first step where for each feature k the optimal update is calculated. We define $BestWt(k, \bar{\alpha})$ as this vector of updates:

$$BestWt(k, \bar{\alpha}) = \arg\min_{\delta} Loss\left(\text{Upd}(\bar{\alpha}, k, \delta)\right)$$

The next step is to calculate the $Loss$ for each feature with its optimal update, which we will call

$$BestLoss(k, \bar{\alpha}) = Loss\left(\text{Upd}(\bar{\alpha}, k, BestWt(k, \bar{\alpha}))\right)$$

Having computed $BestWt$ and $BestLoss$ for each feature, the optimal feature/weight pair can be found:

$$k^* = \arg\min_k BestLoss(k, \bar{\alpha}), \quad \delta^* = BestWt(k^*, \bar{\alpha})$$

The next sections describe how $BestWt$ and $BestLoss$ can be computed for the two loss functions.

2.3.1 Feature Selection for $BoostLoss$

At the first iteration, α_0 is set to optimize $BoostLoss$[5]:

$$\alpha_0 = \arg\min_{\alpha} \sum_i \sum_{j=2}^{n_i} e^{-\left(\alpha\left[L(x_{i,1}) - L(x_{i,j})\right]\right)}$$

Feature selection then proceeds to search for values of the remaining parameters, $\alpha_1 \ldots \alpha_m$.[6]

The first thing to note in the case of $BoostLoss$ is that the updated model involves a simple additive update to the ranking function F:

$$F(x_{i,j}, \text{Upd}(\bar{\alpha}, k, \delta)) = F(x_{i,j}, \alpha) + \delta h_k(x_{i,j})$$

It follows that the margin on example ij also has a simple update:

$$\begin{aligned} & M_{i,j}(\text{Upd}(\bar{\alpha}, k, \delta)) \\ = \; & F(x_{i,1}, \text{Upd}(\bar{\alpha}, k, \delta)) - F(x_{i,j}, \text{Upd}(\bar{\alpha}, k, \delta)) \\ = \; & F(x_{i,1}, \bar{\alpha}) - F(x_{i,j}, \bar{\alpha}) + \delta\left[h_k(x_{i,1}) - h_k(x_{i,j})\right] \\ = \; & M_{i,j}(\bar{\alpha}) + \delta\left[h_k(x_{i,1}) - h_k(x_{i,j})\right] \end{aligned}$$

The updated $BoostLoss$ function can then be written as

$$BoostLoss(\text{Upd}(\bar{\alpha}, k, \delta)) = \sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\text{Upd}(\bar{\alpha},k,\delta))}$$

$$= \sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha}) - \delta\left[h_k(x_{i,1}) - h_k(x_{i,j})\right]}$$

Next, we note that $\left[h_k(x_{i,1}) - h_k(x_{i,j})\right]$ can take on three values: $+1$, -1, or 0. We split the training sample into three sets depending on this value

$$\begin{aligned} A_k^+ &= \{(i,j) : \left[h_k(x_{i,1}) - h_k(x_{i,j})\right] = 1\} \\ A_k^- &= \{(i,j) : \left[h_k(x_{i,1}) - h_k(x_{i,j})\right] = -1\} \\ A_k^0 &= \{(i,j) : \left[h_k(x_{i,1}) - h_k(x_{i,j})\right] = 0\} \end{aligned}$$

We next define W_k^+, W_k^- and W_k^0, which are central to the calculation of $BestWt$ and $BestLoss$:

$$W_k^+ = \sum_{(i,j) \in A_k^+} e^{-M_{i,j}(\bar{\alpha})}$$

There are analogous definitions for W_k^- and W_k^0. $BoostLoss$ is now rewritten in terms of these quantities:

$$\begin{aligned} & BoostLoss(\text{Upd}(\bar{\alpha}, k, \delta)) = \\ & \sum_{(i,j) \in A_k^+} e^{-M_{i,j}(\bar{\alpha}) - \delta} + \sum_{(i,j) \in A_k^-} e^{-M_{i,j}(\bar{\alpha}) + \delta} + \sum_{(i,j) \in A_k^0} e^{-M_{i,j}(\bar{\alpha})} \\ & = e^{-\delta} W_k^+ + e^{\delta} W_k^- + W_k^0 \qquad (2) \end{aligned}$$

[5]We use a simple linear search to find this value

[6]It might be preferable to also allow α_0 to be adjusted as features are added; we leave this to future work.

To find the value of δ that minimizes this loss, we set the differential of (2) w.r.t. δ to 0, giving the following solution:

$$BestWt(k, \bar{\alpha}) = \frac{1}{2} \log \frac{W_k^+}{W_k^-}$$

Plugging this value of δ back into (2) gives the best loss:

$$\begin{aligned} BestLoss(k, \bar{\alpha}) &= 2\sqrt{W_k^+ W_k^-} + W_k^0 \\ &= 2\sqrt{W_k^+ W_k^-} + Z - W_k^+ - W_k^- \\ &= Z - \left(\sqrt{W_k^+} - \sqrt{W_k^-}\right)^2 \qquad (3) \end{aligned}$$

where $Z = \sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})}$ is a constant which appears in the $BestLoss$ for all features, and therefore does not affect their ranking.

As a final point, following Schapire and Singer (1998) we introduce some smoothing to prevent parameter estimates being undefined when either W_k^+ or W_k^- is zero:

$$BestWt(k, \bar{\alpha}) = \frac{1}{2} \log \frac{W_k^+ + \epsilon Z}{W_k^- + \epsilon Z}$$

The smoothing parameter ϵ is chosen using cross-validation.

2.3.2 Feature Selection for $LogLoss$

At the first iteration, α_0 is set to 1. Feature selection then searches for values of the remaining parameters, $\alpha_1 \ldots \alpha_m$. We now describe how to calculate the optimal update for a feature k with the $LogLoss$ loss function. First we recap the definition of the probability of parse $x_{i,q}$ given parameter settings $\bar{\alpha}$:

$$P(x_{i,q}, \bar{\alpha}) = \frac{e^{F(x_{i,q}, \bar{\alpha})}}{\sum_{j=1}^{n_i} e^{F(x_{i,j}, \bar{\alpha})}}$$

Recall that the log-loss is $\sum_i - \log P(x_{i,1}, \bar{\alpha})$.

Unfortunately, unlike the case of $BoostLoss$, an analytic solution to finding $BestWt$ does not exist. However, we can define an iterative solution using techniques from iterative scaling (Della Pietra et al. 1997). We first define $\tilde{h}_k$, the number of times that feature k is seen in the best parse, and $\tilde{p}_k(\bar{\alpha})$, the expected number of times under the model that feature k is seen:

$$\tilde{h}_k = \sum_i h_k(x_{i,1}) \quad \tilde{p}_k(\bar{\alpha}) = \sum_i \sum_{j=1}^{n_i} h_k(x_{i,j}) P(x_{i,j}, \bar{\alpha})$$

Iterative scaling then defines the following update $\tilde{\delta}$

$$\tilde{\delta} = \log \frac{\tilde{h}_k}{\tilde{p}_k(\bar{\alpha})}$$

While in general it is *not* true that $\tilde{\delta} = BestWt(k, \bar{\alpha})$, it *can* be shown that this update leads to an improvement in the $LogLoss$ (i.e., that $LogLoss(\text{Upd}(\bar{\alpha}, k, \tilde{\delta})) \leq LogLoss(\bar{\alpha})$), with equality holding only when α_k is already at the optimal value, in other words when $\arg\min_\delta LogLoss(\text{Upd}(\bar{\alpha}, k, \delta)) = 0$. This suggests the following iterative method for finding $BestWt(k, \bar{\alpha})$:

```
1 Initialization: set $\delta = 0$, $\bar{\alpha}' = \bar{\alpha}$, calculate $\tilde{h}_k$
2 Repeat until convergence of $\delta$:
  – Calculate $\tilde{p}_k(\bar{\alpha}')$
  – $\delta \leftarrow \delta + \log \frac{\tilde{h}_k}{\tilde{p}_k(\bar{\alpha}')}$
  – $\bar{\alpha}' \leftarrow$ Upd$(\bar{\alpha}, k, \delta)$
3 Return $BestWt(k, \bar{\alpha}) = \delta$
```

Once $BestWt(k, \bar{\alpha})$ has been computed, $BestLoss(k, \bar{\alpha})$ can be calculated as $Loss(k, BestWt(k, \bar{\alpha}))$.

2.4 Efficiency Issues

The efficiency of the different algorithms is important in the parsing problem. The training data we eventually used contained around 36,000 sentences, with an average of 27 parses per sentence, giving around 1,000,000 parse trees in total. There were over 500,000 different features.

We first discuss efficiency issues for the boosting algorithm. We describe a "naive" algorithm, but show that a much improved algorithm can be derived, which exploits the fact that only a relatively small number of features are seen on each example. We then discuss the log-likelihood models, also giving an algorithm which exploits the sparseness of the feature space.

Table 1 shows the "naive" boosting algorithm. The calculation of W_k^+ and W_k^- dominates this algorithm; these calculations involve $\sum_{k=1}^{m} |A_k^+| + |A_k^-|$ additions.

Table 2 shows an improved boosting algorithm. The key observation is that when updating the model from $\bar{\alpha}$ to Upd$(\bar{\alpha}, k^*, \delta^*)$ the values W_k^+ and W_k^- remain unchanged for many features, and do not need to be recalculated. In fact, only features which co-occur with k^* on some example must be updated. The algorithm relies on a second pair of indices, for all i, $2 \leq j \leq n_i$ we define

$$\begin{aligned} B_{i,j}^+ &= \{k : [h_k(x_{i,1}) - h_k(x_{i,j})] = 1\} \\ B_{i,j}^- &= \{k : [h_k(x_{i,1}) - h_k(x_{i,j})] = -1\} \qquad (4) \end{aligned}$$

So $B_{i,j}^+$ and $B_{i,j}^+$ are indices from training examples to features. With the algorithm in table 2, updating the values of W_k^+ and W_k^- for the features which co-occur with k^* involves the following number of steps:

$$\sum_{(i,j) \in A_{k^*}^+} |B_{i,j}^+| + |B_{i,j}^-| + \sum_{(i,j) \in A_{k^*}^-} |B_{i,j}^+| + |B_{i,j}^-|$$

We can now compare the expected complexity of these two algorithms. Define A to be the average (over features k) of $|A_k^+| + |A_k^-|$, and B to be the average (over examples i, j) of $|B_{i,j}^+| + |B_{i,j}^-|$. The naive algorithm would run in $O(mA)$ time (m is the number of features), the improved algorithm runs in roughly $O(BA)$ time. While in the worst case $m = B$, in practice the sparseness of the feature space means that B is much smaller than A: in our experiments $m \approx 500,000$ and $B \approx 250$.

Table 1. A naive algorithm for the boosting loss function.

```
Input  Examples $x_{i,j}$ with initial model scores $L(x_{i,j})$, sets
       $A_k^+$, $A_k^-$ for each feature $h_k$, $k = 1 \ldots m$
Initialize  $\bar{\alpha}^0 = \{\alpha_0, 0, 0, \ldots\}$, for all $i$, $2 \leq j \leq n_i$ set
       margins $M_{i,j} = \alpha_0 \left[L(x_{i,1}) - L(x_{i,j})\right]$
Repeat  for $t$ = 1 to $N$
  • for $k$ = 1 to $m$
      – Set $W_k^+ = W_k^- = 0$
      – for $(i,j) \in A_k^+$,   $W_k^+ = W_k^+ + e^{-M_{i,j}}$
      – for $(i,j) \in A_k^-$,   $W_k^- = W_k^- + e^{-M_{i,j}}$
      – $BestWt(k) = \frac{1}{2}\log\frac{W_k^+}{W_k^-}$
      – $BestLoss(k) = 2\sqrt{W_k^+ W_k^-} - W_k^+ - W_k^-$
  • Choose $k^* = \arg\min_k BestLoss(k)$,
           $\delta^* = BestWt(k^*)$
  • for $(i,j) \in A_{k^*}^+$,   $M_{i,j} = M_{i,j} + \delta^*$
  • for $(i,j) \in A_{k^*}^-$,   $M_{i,j} = M_{i,j} - \delta^*$
  • $\bar{\alpha}^t = \text{Upd}(\bar{\alpha}^{t-1}, k^*, \delta^*)$
Output  Final parameter setting $\bar{\alpha}^N$
```

Table 2. An improved algorithm for the boosting loss function.

```
Input  Examples $x_{i,j}$ with initial model scores $L(x_{i,j})$, sets
       $A_k^+$, $A_k^-$, $B_{i,j}^+$, $B_{i,j}^-$,
Initialize
  • $\bar{\alpha}^0 = \{\alpha_0, 0, 0, \ldots\}$
  • Set $M_{i,j} = \alpha_0 \left[L(x_{i,1}) - L(x_{i,j})\right]$.
  • Calculate $W_k^+$, $W_k^-$, $BestWt(k)$ and $BestLoss(k)$
    using the naive algorithm
Repeat  for $t$ = 1 to $N$
  • Choose $k^* = \arg\min_k BestLoss(k)$,
           $\delta^* = BestWt(k^*)$
  • for $(i,j) \in A_{k^*}^+$
      – $\Delta = e^{-M_{i,j}-\delta^*} - e^{-M_{i,j}}$
      – $M_{i,j} = M_{i,j} + \delta^*$
      – for $k \in B_{i,j}^+$,   $W_k^+ = W_k^+ + \Delta$
      – for $k \in B_{i,j}^-$,   $W_k^- = W_k^- + \Delta$
  • for $(i,j) \in A_{k^*}^-$
      – $\Delta = e^{-M_{i,j}+\delta^*} - e^{-M_{i,j}}$
      – $M_{i,j} = M_{i,j} - \delta^*$
      – for $k \in B_{i,j}^+$,   $W_k^+ = W_k^+ + \Delta$
      – for $k \in B_{i,j}^-$,   $W_k^- = W_k^- + \Delta$
  • For all features $k$ whose values of $W_k^+$ and/or $W_k^-$ have
    changed, recalculate $BestWt(k)$ and $BestLoss(k)$
  • $\bar{\alpha}^t = \text{Upd}(\bar{\alpha}^{t-1}, k^*, \delta^*)$
Output  Final parameter setting $\bar{\alpha}^N$
```

We next consider efficiency issues within the $LogLoss$ algorithm. A similar observation can be made, in that when updating the model with a feature/weight pair (k^*, δ^*) many features will have their values for $BestWt$ and $BestLoss$ unchanged. Only those features which co-occur with k^* on some example will need to have their values of $BestWt$ and $BestLoss$ updated. Unfortunately updating these values is much more expensive than in the $BoostLoss$ case. The iterative procedure described in section 2.3.2 must be applied for each feature in turn, and each iteration will involve recalculation of the distribution $\{P(x_{i,1}), P(x_{i,2}) \ldots P(x_{i,n_i})\}$ for each example i on which the feature occurs. This contrasts with the simple updates in the improved boosting algorithm ($W_k^+ = W_k^+ + \Delta$ and $W_k^- = W_k^- + \Delta$). In practice we found that the boosting methods were vastly more efficient than the $LogLoss$ method.

2.5 Related Work

Abney (1997) describes the application of Markov Random Fields to stochastic attribute-value grammars. Della Pietra et al. (1997) describe feature selection methods for MRFs, and Rosenfeld (1997) describes the application of these methods to language modeling for speech recognition. These methods all emphasize the use of MRFs to define a joint probability over the space of all parse trees: the probability of a tree $x_{i,j}$ is

$$P(x_{i,j}) = \frac{e^{F(x_{i,j})}}{\sum_{x \in \mathcal{Z}} e^{F(x)}} \qquad (5)$$

Here $\mathcal{Z}$ is the (infinite) set of possible trees, and the denominator cannot be calculated explicitly. This is a problem for parameter estimation, where an estimate of the denominator is required, and computationally expensive Monte-Carlo methods are required to estimate this value. Notice that (5) is not a direct function of the margins on training examples, and its relation to the error rate is therefore not so clear as in the discriminative approaches described in this paper.

Johnson et al. (1999) suggested training the conditional or pseudo-likelihood of a MRF, the $LogLoss$ function in (1). They point out the computational advantages (faster parameter estimation) over MRFs which define joint probability distributions. Johnson et al. (1999) do not use a feature selection technique, instead using an objective function which includes a gaussian prior on the parameter values, thereby penalizing parameter values which become too large:

$$\bar{\alpha}^* = \arg\min_{\bar{\alpha}} \left(LogLoss(\bar{\alpha}) + \sum_{s=0\ldots m} \frac{\alpha_m^2}{\phi_m^2} \right)$$

Closed-form updates under iterative scaling are not possible with this objective function, instead gradient descent is used to estimate parameter values.

While Della Pietra et al. (1997) focus on joint-probability MRFs, the proofs and techniques they describe carry over quite easily to the conditional probability case. Importantly, they describe feature selection techniques for MRFs. These differ slightly from the methods in this paper, in that at each iteration a new feature is greedily chosen, but then *all* previously chosen features have their values updated to the optimal values. This may be desirable, but we suspect it is computationally expensive, much too expensive for the size of problem we are considering.

Ratnaparkhi (1997) describes the use of maximum entropy techniques applied to parsing. MRF techniques are used to estimate the conditional probabilities $P\left(d_i|\Phi\left(d_1 ... d_{i-1}\right)\right)$ in a history-based parser. Charniak (1999) also describes a method for incorporating additional features in the parser of Charniak (1997). The method gives an impressive improvement over the original model. Both approaches still rely on decomposing a parse tree into a sequence of decisions, and we would argue that the techniques described in this paper, and the MRF models (Della Pietra et al. 1997; Abney 1997; Johnson et al. 1999), have more flexibility in terms of the features that can be included in the model.

The boosting method described in this paper is a special case of the algorithms in Freund et al. (1998); this algorithm is based on the boosting approach to classification in Schapire and Singer (1998). Future work may investigate the use of the more general results in that paper.

3. Experimental Evaluation

We used the Penn Wall Street Journal treebank (Marcus et al. 1993) as training and test data. Sections 2-21 inclusive (around 40,000 sentences) were used as training data, section 23 was used as the final test set. Of the 40,000 training sentences, the first 36,000 were used as training data. The remaining 4,000 sentences were used as development data, and to cross-validate the number of rounds (features) in the model. Model 2 of Collins (1999) was used to parse both the training and test data, producing multiple hypotheses for each sentence.[7] In order to gain a representative set of training data, the 36,000 training sentences were parsed in consecutive 2,000 sentence chunks, each chunk being parsed with a model trained on the remaining 34,000 sentences (this prevented the initial model from being unrealistically "good" on the training sentences). The 4,000 development sentences were parsed with a model trained on the 36,000 training sentences. Section 23 was parsed with a model trained on all 40,000 sentences.

The $BoostLoss$ method was run for 100,000 rounds on the training data. Cross-validation found the optimal stopping point was at 27,083 rounds, at which point 10,084 features had non-zero values (note that the feature selection techniques may result in a given feature being updated more than once). The computation took roughly 1-2 days on an SGI IP25 194 MHZ machine.

[7]A beam search was used to restrict the candidate trees to those whose probability was at least one thousandth of the probability of the highest probability parse.

The $LogLoss$ method was too inefficient to run on the full data set. Instead we made some tests on a smaller subset of the data (5934 sentences, giving 200,000 parse trees and 52,294 features). The boosting method took 40 minutes for 10,000 rounds on this data set. The $LogLoss$ method took 20 hours to complete 3500 rounds (a factor of about 85 times slower). In initial experiments we found $BoostLoss$ to give similar, perhaps slightly better, accuracy than $LogLoss$.

The following features were included in the model (we use an example rule `VP -> PP VBD NP NP SBAR` with head `VBD` for illustration):

Rules All context-free rules in the tree, for example `VP -> PP VBD NP NP SBAR`.

Bigrams Pairs of non-terminals to the left and right of the head of the rule. The example rule would contribute the bigrams `(Right,VP,NP,NP)`, `(Right,VP,NP,SBAR)` and `(Right,VP,SBAR,STOP)` to the right of the head, and `(Left,VP,PP,STOP)` to the left of the head.

Grandparent Rules Same as **Rules**, but also including the non-terminal above the rule.

Grandparent Bigrams Same as **Bigrams**, but also including the non-terminal above the bigrams.

Lexical Bigrams Same as **Bigrams**, but with the lexical heads of the two non-terminals also included.

Two-level Rules Same as **Rules**, but also including the entire rule above the rule.

Two-level Bigrams Same as **Bigrams**, but also including the entire rule above the rule.

Trigrams All trigrams within the rule. The example rule would contribute the trigrams `(VP,STOP,PP,VBD!)`, `(VP,PP,VBD!,NP)`, `(VP,VBD!,NP,NP)`, `(VP,NP,NP,SBAR)` and `(VP,NP,SBAR,STOP)` (`!` is used to mark the head of the rule).

Head-Modifiers All head-modifier pairs, with the grandparent non-terminal also included. The example rule would contribute `(Left,VP,VBD,PP)`, `(Right,VP,VBD,NP)`, `(Right,VP,VBD,NP)`, and `(Right,VP,VBD,SBAR)`.

PPs Lexical trigrams involving the heads of arguments of prepositional phrases. For example in "(NP (NP the president) (PP of (NP the U.S.)" the trigram `(NP,NP,PP,NP,president,of,U.S.)` would be generated, as well as the more general relation `(NP,NP,PP,NP,of,U.S.)`

Distance Head-Modifiers Features involving the distance between head words. For example, assume $dist$ is the number of words between the head words of the `VBD` and `SBAR` in the `(VP,VBD,SBAR)` head-modifier relation in the above rule. This relation would then generate features `(VP,VBD,SBAR,`$dist$`)`, `(VP,VBD,SBAR,`$dist \leq x$`)`

Table 3. Results on Section 23 of the WSJ Treebank. **LR/LP** = labeled recall/precision. **CBs** is the average number of crossing brackets per sentence. **0 CBs,** 2 **CBs** are the percentage of sentences with 0 or ≤ 2 crossing brackets respectively. CH97 = (Charniak 1997), RA97 = (Ratnaparkhi 1997), CH99 = (Charniak 1999), CO99 = (Collins 1999).

MODEL	$\leq$ 40 Words (2245 sentences)				
	LR	LP	CBs	0 CBs	2 CBs
CH97	87.5%	87.4%	1.00	62.1%	86.1%
CO99	88.5%	88.7%	0.92	66.7%	87.1%
CH99	90.1%	90.1%	0.74	70.1%	89.6%
$BoostLoss$	90.1%	90.4%	0.73	70.7%	89.6%
MODEL	**$\leq$ 100 Words (2416 sentences)**				
	LR	LP	CBs	0 CBs	2 CBs
CH97	86.7%	86.6%	1.20	59.5%	83.2%
RA97	86.3%	87.5%	1.21	60.2%	—
CO99	88.1%	88.3%	1.06	64.0%	85.1%
CH99	89.6%	89.5%	0.88	67.6%	87.7%
$BoostLoss$	89.6%	89.9%	0.87	68.3%	87.7%

for all $dist \leq x \leq 9$ and (VP,VBD,SBAR,$dist \geq x$) for all $1 \leq x \leq dist$.

Further Lexicalization In order to generate more features, a second pass was made where all non-terminals were augmented with their lexical heads when these headwords were closed-class words. All features apart from **Head-Modifiers**, **PPs** and **Distance Head-Modifiers** were then generated with these augmented non-terminals.

All of these features were initially generated, but only features seen on at least one parse for at least 5 different sentences were included in the final model (this count cut-off was implemented to keep the number of features down to a tractable number). Table 3 shows results for the method. Collins (1999) was the base model; the $BoostLoss$ model gave a 1.5% absolute improvement over this method. The method gives very similar accuracy to the model of Charniak (1999), which also uses a rich set of initial features in addition to Charniak's original model (Charniak 1997).

4. Conclusions

This paper has considered alternative methods for reranking the output from an initial statistical parser, with a significant improvement in error rate. We feel that we have barely scratched the surface in terms of investigating features that can be incorporated in the model, and look forward to utilizing the flexibility of the approach in incorporating many other types of features in the parsing problem. We also intend to apply the methods to other tasks in NLP or speech recognition.

Acknowledgements

This work has benefited greatly from discussions with Yoram Singer. Thanks also to Steve Abney, Fernando Pereira and the anonymous reviewers for many helpful comments on earlier drafts of this paper.

References

Abney, S. (1997). Stochastic attribute-value grammars. *Computational Linguistics, 23*, 597-618.

Black, E., Jelinek, F., Lafferty, J., Magerman, D., Mercer, R., & Roukos, S. (1992). Towards history-based grammars: using richer models for probabilistic parsing. In *Proceedings of the Fifth DARPA Speech and Natural Language Workshop* (pp. 134–139). San Francisco: Morgan Kaufmann.

Charniak, E. (1997). Statistical parsing with a context-free grammar and word statistics. *Proceedings of the Fourteenth National Conference on Artificial Intelligence* (pp. 598-603). Menlo Park: AAAI Press/MIT Press.

Charniak, E. (1999). *A maximum-entropy-inspired parser* (Technical Report CS99-12). Department of Computer Science, Brown University, Providence, RI.

Collins, M. (1997). Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and 8th Conference of the European Chapter of the Association for Computational Linguistics* (pp. 16–23). San Francisco: Morgan Kaufmann.

Collins, M. (1999). *Head-driven statistical models for natural language parsing*. Doctoral Dissertation, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia.

Della Pietra, S., Della Pietra, V., & Lafferty, J. (1997). Inducing features of random fi elds. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 19*, 380–393.

Freund, Y., Iyer, R.,Schapire, R.E., & Singer, Y. (1998). An effi - cient boosting algorithm for combining preferences. In *Machine Learning: Proceedings of the Fifteenth International Conference*. San Francisco: Morgan Kaufmann.

Friedman, J. H., Hastie, T. and Tibshirani, R. (1998). *Additive logistic regression: a statistical view of boosting*. Unpublished manuscript, Dept. of Statistics, Stanford University, Stanford, CA.

Johnson, M., Geman, S., Canon, S., Chi, S., & Riezler, S. (1999). Estimators for stochastic 'unifi cation-based"grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (pp. 535–541). San Francisco: Morgan Kaufmann.

Marcus, M., Santorini, B., & Marcinkiewicz, M. (1993). Building a large annotated corpus of english: the penn treebank. *Computational Linguistics, 19*, 313-330.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems*. San Francisco: Morgan Kaufmann.

Ratnaparkhi, A. (1997). A linear observed time statistical parser based on maximum entropy models. In *Proceedings of the Second Conference on Empirical Methods in Natural Language Processing*. San Francisco: Morgan Kaufmann.

Rosenfeld, R. (1997). A whole sentence maximum entropy language model. In *Proceedings of the 1997 IEEE Workshop on Speech Recognition and Understanding*. Santa Barbara, California.

Schapire, R.E., & Singer, Y. (1998). Improved boosting algorithms using confi dence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory* (pp. 80–91). San Francisco: Morgan Kaufmann.