Discriminative Reranking for Natural Language Parsing

Michael Collins and Terry Koo
MIT Computer Science and Artificial Intelligence Laboratory*

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 or a generative model which takes these features into account. We introduce a new method for the reranking task, based on the boosting approach to ranking problems described in (Freund et al. 1998). We applied the boosting method to parsing the Wall Street Journal treebank. The method combined the log-likelihood under a baseline model (that of (Collins 1999)) with evidence from an additional 500,000 features over parse trees that were not included in the original model. The new model achieved 89.75% F-measure, a 13% relative decrease in F-measure error over the baseline model's score of 88.2%. The paper also introduces a new algorithm for the boosting approach which takes advantage of the sparsity of the feature space in the parsing data. Experiments show an efficiency gain of a factor of almost 2,700 for the new algorithm over the obvious implementation of the boosting approach. Although the experiments in this paper are on natural language parsing, the approach should be applicable to many other NLP problems which are naturally framed as ranking tasks, for example speech recognition, machine translation, or natural language generation.

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. In the ideal case, the designer of a statistical parser would be able to easily add features to the model that are believed to be useful in discriminating between candidate trees for a sentence. In practice, however, adding new features to a generative model can be awkward – the derivation in the model must be altered to take the new features into account, and this can be an intricate task.

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

* MIT CSAIL, 200 Technology Square, MIT Building NE43, Cambridge, MA 02139

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.

We introduce a new method for the reranking task, based on the boosting approach to ranking problems described in (Freund et al. 1998). The algorithm can be viewed as a feature selection method, optimizing a particular loss function (the exponential loss function) that has been studied in the boosting literature. We applied the boosting method to parsing the Wall Street Journal treebank (Marcus et al. 1993). The method combines the log-likelihood under a baseline model (that of (Collins 1999)) with evidence from an additional 500,000 features over parse trees that were not included in the original model. The baseline model achieved 88.2% F-measure on this task. The new model achieves 89.75% F-measure, a 13% relative decrease in F-measure error.

Although the experiments in this paper are on natural language parsing, the approach should be applicable to many other NLP problems which are naturally framed as ranking tasks, for example speech recognition, machine translation, or natural language generation. See (Collins, 2002a) for an application of the boosting approach to named entity recognition, and (Walker et al., 2001) for the application of boosting techniques for ranking in the context of natural language generation.

The paper also introduces a new, more efficient algorithm for the boosting approach which takes advantage of the sparse nature of the feature space in the parsing data. Other NLP tasks are likely to have similar characteristics in terms of sparsity. Experiments show an efficiency gain of a factor of 2,600 for the new algorithm over the obvious implementation of the boosting approach. Efficiency issues are important, because the parsing task is a fairly large problem, involving around 1 million parse trees, and over 500,000 features. The improved algorithm can perform 100,000 rounds of feature selection on our task in a few hours with current processing speeds. The 100,000 rounds of feature selection take equivalent computation to just over 37 passes over the entire training set (as opposed to 100,000 passes for the "naive" implementation).

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 (Ratnaparkhi, Roukos and Ward 1994; Abney 1997; Della Pietra, Della Pietra and Lafferty 1997; Johnson et al. 1999) has investigated the use of Markov Random Fields (MRFs), or log-linear models as probabilistic models with global features for parsing and other NLP tasks. (Log-linear models are often referred to as maximum-entropy models in the NLP literature.) Similar methods have also been proposed for machine translation (Och and Ney 2002) and language understanding in dialogue systems (Papineni, Roukos and Ward 1997; Papineni, Roukos and Ward 1998). Previous work (Friedman, Hastie and Tibshirani 1998) has drawn connections between log-linear models and boosting for classification problems. One contribution of our research is to draw similar connections between the two approaches to ranking problems. We argue that the efficient boosting algorithm introduced in this paper is an attractive alternative to maximum-entropy models, in particular the feature selection methods of (Ratnaparkhi et al., 1994; Berger, Della Pietra and Della Pietra 1996; Della Pietra et al. 1997; Papineni, Roukos and Ward 1997; Papineni, Roukos and Ward 1998). These maximum-entropy feature selection methods require several full passes over the training set for each round of feature selection, suggesting that at least for the parsing data the improved boosting algorithm is several orders of magnitude more efficient.

The remainder of this paper is structured as follows. Section 2 reviews history-based models for NLP, and highlights the perceived shortcomings of history-based models which motivate the reranking approaches described in the remainder of the paper. Section 3 describes previous work (Friedman et al. 1998; Duffy and Helmbold 1999; Mason et. al 1999; Lebanon and Lafferty 2002; Collins, Schapire and Singer, 2002) that derives connections between boosting and maximum-entropy models for the simpler case of classification problems – this work will form the basis for the reranking methods. Section 4 describes how these approaches can be generalized to ranking problems. We introduce loss functions for boosting and MRF approaches, and discuss optimization methods. We also derive the efficient algorithm for boosting in this section. Section 5 discusses related work in more detail. Section 6 gives experimental results, investigating the performance improvements on parsing, efficiency issues, and the effect of various parameters of the boosting algorithm. Finally, section 7 gives conclusions.

The reranking models in this paper were originally introduced in (Collins, 2000). In this paper we give considerably more detail in terms of the algorithms involved, their justification, and their performance in experiments on natural language parsing.

2 History-Based Models

Before discussing the reranking approaches, we will describe *history-based* models (Black et al. 1992b). They are important for a few reasons. First, several of the best performing parsers on the WSJ treebank (Ratnaparkhi 1997, Charniak 1997, 2000, Collins 1997, 1999) are 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 $f : \mathcal{X} \to \mathcal{Y}$ given training examples $\langle x_i, y_i \rangle$ where $x_i \in \mathcal{X},\ y_i \in \mathcal{Y}$. We define $\mathrm{GEN}(x) \subset \mathcal{Y}$ to be the set of candidates for a given input x. In the parsing problem x is a sentence, and $\mathrm{GEN}(x)$ is a set of candidate trees for that sentence. A particular characteristic of the problem is the complexity of $\mathrm{GEN}(x)$: $\mathrm{GEN}(x)$ can be very large, and each member of $\mathrm{GEN}(x)$ has rich internal structure. This contrasts with "typical" classification problems where $\mathrm{GEN}(x)$ 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.[1] The most likely parse for each sentence x is then $\arg\max_{y \in \mathrm{GEN}(x)} P(y, x)$. 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 $\langle d_1 ... d_n \rangle$. The sequence $\langle d_1 ... d_n \rangle$ 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

$$P(y, x) \quad = \quad \prod_{i=1...n} P\left(d_i | \Phi\left(d_1 ... d_{i-1}\right)\right)$$

Here, $(d_1 ... d_{i-1})$ is the *history* for the i'th decision. Φ is a function which groups histories

1 To be more precise, generative probabilistic models assign joint probabilities $P(y, x)$ to each (y, x) pair. Similar arguments apply to conditional history-based models, which define conditional probabilities $P(y \mid x)$ through a definition

$$P(y \mid x) \quad = \quad \prod_{i=1...n} P\left(d_i | \Phi\left(d_1 ... d_{i-1}, x\right)\right)$$

where $d_1 \ldots d_n$ are again the decisions made in building a parse, and Φ is a function that groups histories into equivalence classes. Note that x is added to the domain of Φ (the context on which decisions are conditioned). See (Ratnaparkhi 1997) for one example of a method using this approach.

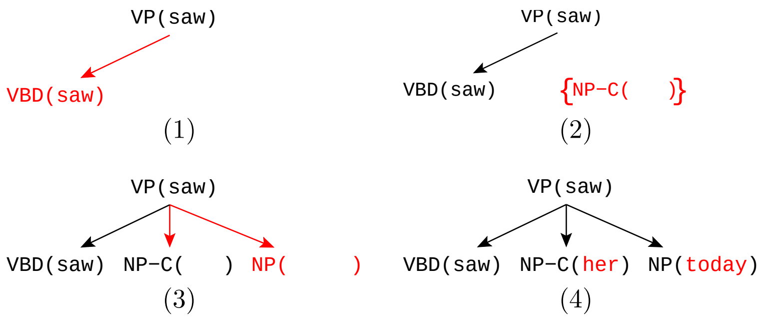


Figure 1
The sequence of decisions involved in generating the right hand side of a lexical rule.

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 $\langle d_1 ... d_n \rangle$ 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 non-terminal being expanded (i.e., 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 rule expansions $\langle \gamma_i \rightarrow \beta_i \rangle$ 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, as follows:

- 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*).

Figure 1 illustrates this process. Each of the above 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_k multiplied by features h_k. Each feature $h_k(x, y)$ is the count of a different "event", or fragment within the tree. As an example, consider a PCFG with rules $\langle \gamma_k \rightarrow \beta_k \rangle$ for $1 \leq k \leq m$. If $h_k(x, y)$ is the number of times $\langle \gamma_k \rightarrow \beta_k \rangle$ is seen in the tree, and $\alpha_k = \log P(\beta_k | \gamma_k)$ is the parameter associated with that rule, then

$$\log P(y, x) = \sum_{k=1}^{m} \alpha_k h_k(x, y)$$

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_k define an m-dimensional vector of counts which represent the tree. The parameters α_k 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 88), 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_k, 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. 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. In contrast, it would be trivial to implement a feature $h_k(x, y)$ which is 1 if y contains a verb, 0 otherwise.

3 Logistic Regression and Boosting

We now turn to machine learning methods for the ranking task. In this section we review two methods for binary classification problems: logistic regression (or maximum-entropy) models, and boosting. These methods will form the basis for the reranking approaches described in later sections of the paper. Maximum-entropy models are a very popular method within the computational linguistics community, see for example (Berger, Della Pietra and Della Pietra 1996) for an early article which introduces the models and motivates them. Boosting approaches to classification have received considerable attention in the machine learning community, since the introduction of AdaBoost by (Freund and Schapire, 1997).

Boosting algorithms, and in particular the relationship between boosting algorithms and maximum-entropy models, are perhaps not familiar topics in the NLP literature. However there has recently been much work drawing connections between the two methods (Friedman et al. 1998; Lafferty 1999; Duffy and Helmbold 1999; Mason et. al 1999; Lebanon and Lafferty 2002; Collins, Schapire and Singer, 2002); in this section we give a review of this work. Much of this work has focused on binary classification problems, and this section is also restricted to problems of this type. Later in the paper we show how several of the ideas can be carried across to reranking approaches.

3.1 Binary Classification Problems

The general set-up for binary classification problems is as follows:

- The "input domain" (set of possible inputs) is $\mathcal{X}$.
- The "output domain" (set of possible labels) is simply a set of two labels, $\mathcal{Y} = \{-1, +1\}$.[2]

2 It turns out to be convenient to define $\mathcal{Y} = \{-1, +1\}$ rather than $\mathcal{Y} = \{0, +1\}$, for example.

- The training set is an array of n labelled examples, $\langle (x_1, y_1), (x_2, y_2) \dots (x_n, y_n) \rangle$ where each $x_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$.

- Input examples are represented through m "features", which are functions $h_k : \mathcal{X} \rightarrow \Re$ for $k = 1 \dots m$. It is also sometimes convenient to think of an example x as being represented by an m-dimensional "feature vector" $\phi(x) = \langle h_1(x), h_2(x), \dots, h_m(x) \rangle$.

- Finally, there is a parameter vector, $\bar{\alpha} = \langle \alpha_1 \dots \alpha_m \rangle$, where each $\alpha_k \in \Re$, hence $\bar{\alpha}$ is an m-dimensional real-valued vector.

We will see that both logistic regression and boosting implement a linear, or hyperplane, classifier. This means that given an input example x, and parameter values $\bar{\alpha}$, the output from the classifier is

$$\text{sign}\left(F(x, \bar{\alpha})\right) \tag{1}$$

where

$$F(x, \bar{\alpha}) = \sum_{k=1}^{m} \alpha_k h_k(x) = \bar{\alpha} \cdot \phi(x) \tag{2}$$

Here $\bar{\alpha} \cdot \phi(x)$ is the inner or dot product between the vectors $\bar{\alpha}$ and $\phi(x)$, and $\text{sign}(z) = 1$ if $z \geq 0$, $\text{sign}(z) = -1$ otherwise. Geometrically, the examples x are represented as vectors $\phi(x)$ in some m-dimensional vector space, and the parameters $\bar{\alpha}$ define a hyperplane which passes through the origin[3] of the space, and has $\bar{\alpha}$ as its normal. Points lying on one side of this hyperplane are classified as $+1$, points on the other side are classified as -1. The central question in learning is how to set the parameters $\bar{\alpha}$, given the training examples $\langle (x_1, y_1), (x_2, y_2) \dots (x_n, y_n) \rangle$. Logistic regression and boosting involve different algorithms and criteria for training the parameters $\bar{\alpha}$, but recent work (Friedman et al. 1998; Duffy and Helmbold 1999; Mason et. al 1999; Lebanon and Lafferty 2002; Collins, Schapire and Singer, 2002) has shown that the methods have strong similarities. The next section describes parameter estimation methods.

3.2 Loss Functions for Logistic Regression and Boosting

A central idea in both logistic-regression and boosting is that of a loss function, which drives the parameter estimation methods of the two approaches. This section describes loss functions for binary classification. Later in the paper, we introduce loss functions for reranking tasks which are closely related to the loss functions for classification tasks.

First, consider a logistic regression model. The parameters of the model $\bar{\alpha}$ are used to define a conditional probability

$$P(y \mid x, \bar{\alpha}) = \frac{e^{yF(x,\bar{\alpha})}}{1 + e^{yF(x,\bar{\alpha})}} \tag{3}$$

where $F(x, \bar{\alpha})$ is as defined in Equation (2). Some form of maximum-likelihood estimation is often used for parameter estimation. The parameters are chosen to maximize the log-likelihood of the training set; equivalently, we will (to emphasize the similarities to the

3 It might seem to be a restriction to have the hyperplane passing through the origin of the space. However if a constant "bias" feature $h_{m+1}(x) = 1$ for all x is added to the representation, a hyperplane passing through the origin in this new space is equivalent to a hyperplane in general position in the original m-dimensional space.

boosting approach) talk about minimizing the negative log-likelihood. The negative log-likelihood, LogLoss($\bar{\alpha}$), is defined as

$$\begin{aligned}\text{LogLoss}(\bar{\alpha}) = -\sum_{i=1}^{n} \log P(y_i \mid x_i, \bar{\alpha}) &= -\sum_{i=1}^{n} \log \left(\frac{e^{y_i F(x_i, \bar{\alpha})}}{1 + e^{y_i F(x_i, \bar{\alpha})}} \right) \\ &= \sum_{i=1}^{n} \log \left(1 + e^{-y_i F(x_i, \bar{\alpha})} \right) \qquad (4)\end{aligned}$$

There are many methods in the literature for minimizing LogLoss($\bar{\alpha}$) with respect to $\bar{\alpha}$, for example Generalized or Improved Iterative Scaling (Berger, Della Pietra and Della Pietra 1996; Della Pietra et al. 1997), or conjugate gradient methods (Malouf 2002). In the next section we describe feature selection methods, as described in (Berger, Della Pietra and Della Pietra 1996; Della Pietra et al. 1997).

Once the parameters $\bar{\alpha}$ are estimated on training examples, the output for an example x is the most likely label under the model,

$$\arg\max_{y \in \mathcal{Y}} P(y \mid x, \bar{\alpha}) = \arg \max_{y \in \{-1,+1\}} yF(x, \bar{\alpha}) = \text{sign}\left(F(x, \bar{\alpha})\right) \qquad (5)$$

where as before, sign(z) $= 1$ if $z \geq 0$, sign(z) $= -1$ otherwise. Thus we see that the logistic regression model implements a hyperplane classifier.

In boosting, a different loss function is used, namely ExpLoss($\bar{\alpha}$), which is defined as

$$\text{ExpLoss}(\bar{\alpha}) = \sum_{i=1}^{n} e^{-y_i F(x_i, \bar{\alpha})} \qquad (6)$$

This loss function is minimized using a feature selection method, which we will describe in the next section.

There are strong similarities between LogLoss (equation 4) and ExpLoss (equation 6). In making connections between the two functions, it useful to consider a third function of the parameters and training examples,

$$\text{Error}(\bar{\alpha}) = \sum_{i=1}^{n} [\![y_i F(x_i, \bar{\alpha}) \leq 0]\!] \qquad (7)$$

where $[\![\pi]\!]$ is 1 if π is true, 0 otherwise. Error($\bar{\alpha}$) is the number of incorrectly classified training examples under parameter values $\bar{\alpha}$.

Finally, it will be useful to define the *margin* on the i'th training example, given parameter values $\bar{\alpha}$, as

$$M_i(\bar{\alpha}) = y_i F(x_i, \bar{\alpha}) \qquad (8)$$

With these definitions, the three loss functions can be written in the following form:

$$\begin{aligned}\text{LogLoss}(\bar{\alpha}) &= \sum_{i=1}^{n} f(M_i(\bar{\alpha})) \quad \text{where } f(z) = \log\left(1 + e^{-z}\right) \\ \text{ExpLoss}(\bar{\alpha}) &= \sum_{i=1}^{n} f(M_i(\bar{\alpha})) \quad \text{where } f(z) = e^{-z} \\ \text{Error}(\bar{\alpha}) &= \sum_{i=1}^{n} f(M_i(\bar{\alpha})) \quad \text{where } f(z) = [\![z \leq 0]\!]\end{aligned}$$

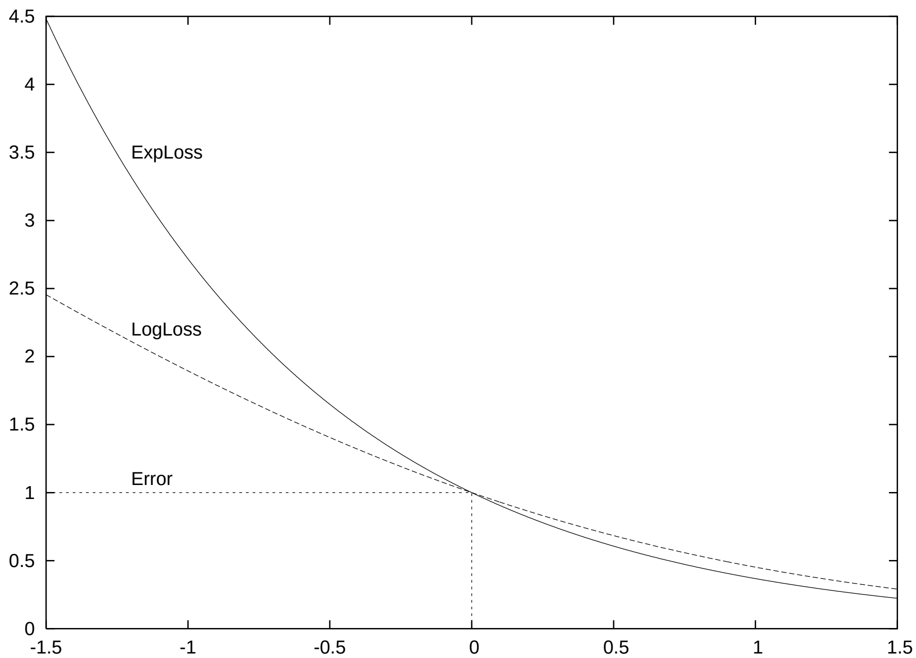


Figure 2
Potential functions underlying ExpLoss, LogLoss, and Error. The graph labelled ExpLoss is a plot of $f(z) = e^{-z}$ for $z = [-1.5 \ldots 1.5]$; LogLoss shows a similar plot for $f(z) = \log\left(1 + e^{-z}\right)$; Error is a plot of $f(z) = [\![z \leq 0]\!]$.

The three loss functions differ only in their choice of an underlying "potential function" of the margins, $f(z)$. This function is $f(z) = \log(1 + e^{-z})$, $f(z) = e^{-z}$, or $f(z) = [\![z \leq 0]\!]$ for LogLoss, ExpLoss and Error respectively. The $f(z)$ functions penalize negative margins on training examples. The simplest function, $f(z) = [\![z \leq 0]\!]$, gives a cost of 1 if a margin is negative (an error is made), 0 otherwise. ExpLoss and LogLoss involve definitions for $f(z)$ which quickly tend to 0 as $z \to \infty$, but which heavily penalize increasingly negative margins.

Figure 2 shows plots for the three definitions of $f(z)$. The functions $f(z) = e^{-z}$ and $f(z) = \log(1 + e^{-z})$ are both upper bounds on the error function, so that minimizing either LogLoss or ExpLoss can be seen as minimizing an upper bound on the number of training errors. (Note that minimizing Error($\bar{\alpha}$) itself is known to be at least NP-hard if no parameter settings can achieve zero errors on the training set, see for example (Hoffgen, van Horn, and Simon 1995).) As $z \to \infty$, the functions $f(z) = e^{-z}$ and $f(z) = \log(1 + e^{-z})$ become increasingly similar, because $\log(1 + e^{-z}) \to e^{-z}$ as $e^{-z} \to 0$. For negative z, the two functions behave quite differently. $f(z) = e^{-z}$ shows an exponentially growing cost function as $z \to -\infty$. In contrast, as $z \to -\infty$ it can be seen that $\log(1 + e^{-z}) \to \log(e^{-z}) = -z$, so this function shows asymptotically linear growth for negative z. As a final remark, note that both $f(z) = e^{-z}$ and $f(z) = \log(1 + e^{-z})$ are convex in z, with the result that LogLoss($\bar{\alpha}$) and ExpLoss($\bar{\alpha}$) are convex in the parameters $\bar{\alpha}$. This means that there are no problems with local minima when optimizing these two loss functions.

3.3 Feature Selection Methods

In this paper we concentrate on feature-selection methods – algorithms which aim to make progress in minimizing the loss functions LogLoss($\bar{\alpha}$) and ExpLoss($\bar{\alpha}$) while using a small number of features (equivalently, ensuring that most parameter values in $\bar{\alpha}$ are zero). Roughly speaking, the motivation for using a small number of features is the hope that this will prevent overfitting in the models.

Feature selection methods have been proposed in the maximum-entropy literature by several authors (Ratnaparkhi et al., 1994; Berger, Della Pietra and Della Pietra 1996; Della Pietra et al. 1997; Papineni, Roukos and Ward 1997; Papineni, Roukos and Ward 1998). For example, (Ratnaparkhi et al., 1994; Berger, Della Pietra and Della Pietra 1996) suggest the following method:

Step 1 Throughout the algorithm, maintain a set of active features. Initialize this set to be empty.

Step 2 Choose a feature from outside of the set of active features which has largest estimated impact in terms of reducing the loss function LogLoss, and add this to the active feature set.

Step 3 Minimize LogLoss($\bar{\alpha}$) with respect to the set of active features – that is, allow only the active features to take non-zero parameter values when minimizing LogLoss. Return to **Step 2**.

Methods in the boosting literature (see for example (Schapire and Singer 1998)) can be considered to be feature selection methods of the following form:

Step 1 Start with all parameter values set to zero.

Step 2 Choose a feature which has largest estimated impact in terms of reducing the loss function ExpLoss.

Step 3 Update the parameter for the feature chosen at **Step 2** in such a way as to minimize the ExpLoss($\bar{\alpha}$) with respect to this one parameter. All other parameter values are left fixed. Return to **Step 2**.

The difference with this latter "boosting" approach is that in Step 3, only one parameter value is adjusted – namely, the parameter corresponding to the newly chosen feature. Note that in this framework, the same feature may be chosen at more than one iteration[4]. The maximum-entropy feature selection method can be quite inefficient, as the entire model is updated at each step. For example, (Ratnaparkhi 1998) quotes times of around 30 hours for 500 rounds of feature selection on a prepositional phrase attachment task. These experiments were performed in 1998, when processors were no doubt considerably slower than machines today. However the PP attachment task is much smaller than the parsing task that we are addressing: our task involves around 1,000,000 examples, with perhaps a few hundred features per example, and 100,000 rounds of feature selection; this compares to 20,000 examples, 16 features per example, and 500 rounds of feature selection for the PP attachment task in (Ratnaparkhi 1998). As a rough estimate, assuming that computational complexity scales linearly in these factors, our task

4 i.e., The feature may be repeatedly updated, although the same feature will never be chosen in consecutive iterations, because after an update the model is minimized with respect to the selected feature.

is $\frac{1,000,000}{20,000} \times \frac{320}{16} \times \frac{100,000}{500} = 200,000$ as large as the PP attachment task. These figures suggest that the maximum-entropy feature selection approach may be infeasible for large-scale tasks such as the one in this paper.

The fact that the boosting approach does not update the entire model at each round of feature selection may be a disadvantage in terms of the number of features, or test data accuracy, of the final model. There is reason for concern that Step 2 will at some iterations mistakenly choose features which are apparently useful in reducing the loss function, but which would have little utility if the entire model had been optimized at the previous iteration of Step 3. However, previous empirical results for boosting have shown that it is a highly effective learning method, suggesting that this is not in fact a problem for the approach. Given the previous strong results for the boosting approach, and for reasons of computational efficiency, we pursue the boosting approach to feature selection in this paper.

3.4 Statistical Justification for the Methods

Minimization of LogLoss is most often justified as a parametric, maximum-likelihood (ML) approach to estimation. Thus this approach benefits from the usual guarantees for ML estimation – if the distribution generating examples is within the class of distributions specified by the log-linear form, then in the limit as the sample size goes to infinity the model will be optimal in the sense of convergence to the true underlying distribution generating examples. As far as we are aware, behaviour of the models for *finite* sample sizes is less well understood. In particular, while feature selection methods have often been proposed, little theoretical justification has been given for them. It seems intuitive that a model with a smaller number of parameters will require fewer samples for convergence, but this is not necessarily the case, and at present this intuition lacks a theoretical basis. Feature selection methods can probably be motivated from either a Bayesian perspective (through a prior favouring models with a smaller number of non-zero parameters), or from a frequentist/goodness-of-fit perspective (models with fewer parameters are less likely to fit the data by chance), but this requires additional research.

The statistical justification for boosting approaches is quite different. Boosting algorithms were originally developed within the PAC (Valiant, 1984) framework for machine learning, specifically to address questions regarding the equivalence of weak and strong learning. (Freund and Schapire, 1997) originally introduced AdaBoost, and gave a first set of statistical guarantees for the algorithm. (Schapire, Freund, Bartlett, and Lee 1998) gives a second set of guarantees based on the analysis of margins on training examples. Both papers assume that a fixed distribution $D(x, y)$ is generating both training and test examples, and that the goal is to find a hypothesis with a small number of expected errors with respect to this distribution. The form of the distribution is not assumed to be known, and in this sense the guarantees are non-parametric, or "distribution free". (Freund and Schapire, 1997) show that if the weak learning assumption holds (i.e., roughly speaking, a feature with error rate better than chance can be found for any distribution over the sample space $\mathcal{X} \times \{-1, +1\}$), then the training error for the ExpLoss method decreases rapidly enough for there to be good generalization to test examples. (Schapire, Freund, Bartlett, and Lee 1998) show that under the same assumption, minimization of ExpLoss using the feature selection method ensures that the distribution of margins on training data develops in such a way that good generalization performance on test examples is guaranteed.

4 Reranking Approaches

This section describes how the ideas from classification problems can be extended to reranking tasks. A baseline statistical parser is used to generate N-best output for both its training set, and for test data sentences. Each candidate parse for a sentence is represented as a feature vector which includes the log-likelihood under the baseline model, as well as a large number of additional features. The additional features can in principle be any predicates over sentence/tree pairs. Evidence from the initial log-likelihood and the additional features is combined using a linear model. Parameter estimation becomes a problem of learning how to combine these different sources of information. The boosting algorithm we use is related to the generalization of boosting methods to ranking problems in (Freund et al. 1998); we also introduce an approach related to the conditional Markov Random Field approaches of (Ratnaparkhi et al., 1994; Papineni, Roukos and Ward 1997; Papineni, Roukos and Ward 1998; Johnson et al. 1999; Och and Ney 2002).

Section 4.1 gives a formal definition of the reranking problem. Section 4.2 introduces loss functions for reranking that are analogous to the LogLoss and ExpLoss functions in section 3.2. Section 4.3 describes a general approach to feature selection methods with these loss functions. Section 4.4 describes a first algorithm for the exponential (ExpLoss) loss function; section 4.5 introduces a more efficient algorithm for the case of ExpLoss. Finally, section 4.6 describes issues in feature selection algorithms for the LogLoss loss function.

4.1 Problem Definition

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

- s_i is the i'th sentence in the training set. There are n sentences in training data, so that $1 \leq i \leq n$.

- $x_{i,j}$ is the j'th parse of the i'th sentence.

  There are n_i parses for the i'th sentence, so that $1 \leq i \leq n$ 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 pair $\langle s_i, t_{i,j} \rangle$ where s_i is the i'th sentence in training data, and $t_{i,j}$ is the j'th tree for this sentence).

- $Score(x_{i,j})$ is the "score" for parse $x_{i,j}$, a measure of the similarity of $x_{i,j}$ to the gold-standard parse. For example $Score(x_{i,j})$ might be the F-measure accuracy of parse $x_{i,j}$ compared to the gold-standard parse for s_i.

- Without loss of generality, we assume $x_{i,1}$ to be the highest scoring parse for the i'th sentence.[5] 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 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.

5 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 $x_{i,1}$.

We will represent candidate parse trees through m features, h_k for $k = 1 \ldots m$. Each h_k is an indicator function, for example

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

We will see that the restriction to binary valued features is important for the simplicity and efficiency of the algorithms.[6] We also assume a vector of $m+1$ parameters, $\bar{\alpha} = \{\alpha_0, \alpha_1 \ldots \alpha_m\}$. The *ranking function* for a parse tree x implied by a parameter vector $\bar{\alpha}$ is defined as

$$F(x, \bar{\alpha}) = \alpha_0 L(x) + \sum_{k=1}^{m} \alpha_k h_k(x)$$

Given a new test sentence s, with parses x_j for $j = 1 \ldots N$, the output of the model will be the highest scoring tree under the ranking function

$$\arg \max_{x \in \{x_1 \ldots x_N\}} F(x, \bar{\alpha})$$

Thus $F(x, \bar{\alpha})$ can be interpreted as a measure of how plausible a parse x is, higher scores meaning that x is more plausible. Competing parses for the same sentence are ranked in order of plausibility by this function. We can recover the base ranking function – the log-likelihood $L(x)$ – by setting α_0 to a positive constant, and setting all other parameter values to be 0. Our intention is to use the training examples to pick parameter values which improve upon this initial ranking.

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 feature selection methods for the different loss functions.

4.2 Loss Functions for Ranking Problems

4.2.1 Ranking Errors, and Margins 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:

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

where again, $[\![\pi]\!]$ is 1 if π is true, 0 otherwise. In the ranking problem we define the *margin* for each example $x_{i,j}$ such that $i = 1 \ldots n$, $j = 2 \ldots n_i$ as

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

Thus M_{ij} is the difference in ranking score between the correct parse of a sentence, and a competing parse $x_{i,j}$. It follows that

$$\text{Error}(\bar{\alpha}) = \sum_i \sum_{j=2}^{n_i} [\![M_{ij}(\bar{\alpha}) \leq 0]\!]$$

6 Note that features tracking the *counts* of different rules can be simulated through several features which take value 1 if a rule is seen ≥ 1 time, ≥ 2 times, ≥ 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.

4.2.2 Log-Likelihood The first loss function is that suggested by Markov Random Fields. As suggested by (Ratnaparkhi et al., 1994; Johnson et al. 1999), the conditional probability of $x_{i,q}$ being the correct parse for the i'th sentence is defined as

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

Given a new test sentence s, with parses x_j for $j = 1 \ldots N$, the most likely tree is

$$\arg\max_{x_j} \frac{e^{F(x_j,\bar{\alpha})}}{\sum_{q=1}^{N} e^{F(x_q,\bar{\alpha})}} = \arg\max_{x_j} F(x_j, \bar{\alpha})$$

Hence once the parameters are trained, the ranking function is used to order candidate trees for test examples.

The log-likelihood of the training data is

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

Under maximum likelihood estimation, the parameters $\bar{\alpha}$ would be set to maximize the log-likelihood. Equivalently, we will again talk about minimizing the negative log-likelihood. Some manipulation shows that the negative log-loss is a function of the margins on training data:

$$\begin{aligned}
\text{LogLoss}(\bar{\alpha}) &= \sum_i -\log \frac{e^{F(x_{i,1},\bar{\alpha})}}{\sum_{j=1}^{n_i} e^{F(x_{i,j},\bar{\alpha})}} \\
&= \sum_i -\log \frac{1}{\sum_{j=1}^{n_i} e^{-(F(x_{i,1},\bar{\alpha})-F(x_{i,j},\bar{\alpha}))}} \\
&= \sum_i \log\left(1 + \sum_{j=2}^{n_i} e^{-(F(x_{i,1},\bar{\alpha})-F(x_{i,j},\bar{\alpha}))}\right) \\
&= \sum_i \log\left(1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})}\right) \qquad (9)
\end{aligned}$$

Note the similarity of Equation (9) to the LogLoss function for classification in Equation (4).

4.2.3 Exponential Loss The next loss function is based on the boosting method described in (Schapire and Singer 1998). It is a special case of the general ranking methods described in (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:

$$\text{ExpLoss}(\bar{\alpha}) = \sum_i \sum_{j=2}^{n_i} e^{-(F(x_{i,1},\bar{\alpha})-F(x_{i,j},\bar{\alpha}))} = \sum_i \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})} \qquad (10)$$

Note the similarity of Equation (10) to the LogLoss function for classification in Equation (6). It can be shown that $\text{ExpLoss}(\bar{\alpha}) \geq \text{Error}(\bar{\alpha})$, so that minimizing $\text{ExpLoss}(\bar{\alpha})$

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

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

We generalize the ExpLoss function slightly, by allowing a weight for each example $x_{i,j}$, for $i = 1 \ldots n, j = 2 \ldots n_i$. We will use $S_{i,j}$ to refer to this weight. In particular, in some experiments in this paper we will use the following definition:

$$S_{i,j} = Score(x_{i,1}) - Score(x_{i,j}) \tag{11}$$

where, as defined in section 4.1, $Score(x_{i,j})$ is some measure of the “goodness” of a parse, such as F-measure (see Section 6 for the exact definition of $Score$ used in our experiments). The definition for ExpLoss is modified to be

$$\text{ExpLoss}(\bar{\alpha}) = \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-M_{i,j}(\bar{\alpha})}$$

This definition now takes into account the importance, $S_{i,j}$, of each example. It is an upper bound on the following quantity

$$\sum_i \sum_{j=2}^{n_i} S_{i,j} [\![M_{i,j} < 0]\!]$$

which is the number of errors weighted by the factors $S_{i,j}$. The original definition of ExpLoss in Equation (10) can be recovered by setting $S_{i,j} = 1$ for all i, j (i.e., by giving equal weight to all examples). In our experiments we found that a definition of $S_{i,j}$ such as that in Equation (11) gave improved performance on development data, presumably because it takes into account the relative cost of different ranking errors in training data examples.

4.3 A General Approach to Feature Selection

At this point we have definitions for ExpLoss and LogLoss which are analogous to the definitions in section 3.1 for binary classification tasks. Section 3.3 introduced the idea of feature selection methods; the current section gives a more concrete description of the methods used in our experiments.

The goal of feature selection methods is to find a small subset of the features that contribute most to reducing the loss function. The methods we consider are greedy, at each iteration picking the feature h_k 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 $\text{Upd}(\bar{\alpha}, k, \delta)$ 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):

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

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

Note that this is essentially the idea behind the "boosting" approach to feature selection introduced in section 3.3. In contrast, the feature selection method of (Berger, Della Pietra and Della Pietra 1996), also described in section 3.3, would involve updating parameter values for *all* selected features at step 2(b).

The main computation for both loss functions involves search for the optimal feature/weight pair (k^*, δ^*). In both cases our approach to solving this problem is a two-step approach. In the first step the optimal update for each feature k is calculated. We define $BestWt(k, \bar{\alpha})$ as the optimal update for the k'th feature (it must be calculated for all features $k = 1 \ldots m$):

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

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

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

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.

4.4 Feature Selection for ExpLoss

At the first iteration, α_0 is set to optimize ExpLoss (recall that $L(x_{i,j})$ is the log-likelihood for parse $x_{i,j}$ under the base parsing model):

$$\alpha_0 = \arg\min_{\alpha} \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-(\alpha[L(x_{i,1}) - L(x_{i,j})])} \tag{12}$$

In initial experiments we found that this step was crucial to the performance of the method (as opposed to simply setting $\alpha_0 = 1$, for example). It ensures that the contribution of the log-likelihood feature is well-calibrated with respect to the exponential loss function. In our implementation α_0 was optimized using simple brute-force search. All values of α_0 between 0.001 and 10 at increments of 0.001 were tested, and the value which minimized the function in Equation (12) was chosen.

Feature selection then proceeds to search for values of the remaining parameters, $\alpha_1 \ldots \alpha_m$. (Note that it might be preferable to also allow α_0 to be adjusted as features are added; we leave this to future work.) This requires calculation of the terms $BestWt(k, \bar{\alpha})$

and $BestLoss(k, \bar{\alpha})$ for each feature. For binary-valued features these values have closed-form solutions, which is computationally very convenient. We now describe the form of these updates. See Appendix A for how the updates can be derived (the derivation is essentially the same as that in (Schapire and Singer 1998)).

First, we note that for any feature, assuming binary-valued feature values, that $[h_k(x_{i,1}) - h_k(x_{i,j})]$ can take on three values: $+1$, -1, or 0. For each k we define the following sets:

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

Thus A_k^+ is the set of training examples where the k'th feature is seen in the correct parse but not in the competing parse; A_k^- is the set where the k'th feature is seen in the incorrect but not the correct parse.

Based on these definitions, we next define W_k^+ and W_k^- as follows:

$$
W_k^+ = \sum_{(i,j) \in A_k^+} S_{i,j} e^{-M_{i,j}(\bar{\alpha})} \tag{13}
$$

$$
W_k^- = \sum_{(i,j) \in A_k^-} S_{i,j} e^{-M_{i,j}(\bar{\alpha})} \tag{14}
$$

Given these definitions, it can be shown (see Appendix A) that

$$
BestWt(k, \bar{\alpha}) = \frac{1}{2} \log \frac{W_k^+}{W_k^-} \tag{15}
$$

and

$$
BestLoss(k, \bar{\alpha}) = Z - \left(\sqrt{W_k^+} - \sqrt{W_k^-} \right)^2 \tag{16}
$$

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

As (Schapire and Singer 1998) point out, the updates in Equation (15) can be problematic, as they are undefined (infinite) when either W_k^+ or W_k^- is zero. Following (Schapire and Singer 1998), we introduce smoothing through a parameter ϵ and the following new definition of $BestWt$:

$$
BestWt(k, \bar{\alpha}) = \frac{1}{2} \log \frac{W_k^+ + \epsilon Z}{W_k^- + \epsilon Z} \tag{17}
$$

The smoothing parameter ϵ is chosen through optimization on a development set.

See figure 3 for a direct implementation of the feature selection method for ExpLoss. We use an array of values

$$
G_k = \left| \sqrt{W_k^+} - \sqrt{W_k^-} \right|
$$

to indicate the *gain* of each feature (i.e. the impact that choosing this feature will have on the ExpLoss function). The features are ranked by this quantity. It can be seen that almost all of the computation involves the calculation of Z, and W_k^+ and W_k^- for each feature k. Once these values have been computed, the optimal feature and its update can be chosen.

Input

- Examples $x_{i,j}$ for $i = 1 \dots n$, $j = 1 \dots n_i$ drawn from some set $\mathcal{X}$.
- Weights $S_{i,j}$ representing importance of examples.
- Initial model log-likelihoods $L(x_{i,j})$, for all examples $x_{i,j}$.
- Feature functions $h_k : \mathcal{X} \rightarrow \{0, 1\}$ for $k = 1 \dots m$.
- Smoothing parameter ϵ (usually chosen by optimization on development data).
- Number of rounds N (usually chosen by optimization on development data).

Initialize

- Set $\alpha_0 = \arg\min_\alpha \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-\alpha[L(x_{i,1}) - L(x_{i,j})]}$.
- Set $\alpha_k = 0$ for $k = 1 \dots m$.
- For all i, $2 \leq j \leq n_i$ set margins $M_{i,j} = \alpha_0 \, [L(x_{i,1}) - L(x_{i,j})]$.
- For all $k = 1 \dots m$, set

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

Repeat for $t = 1$ to N

- Calculate $Z = \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-M_{i,j}}$
- for $k = 1$ to m

  — Set $W_k^+ = W_k^- = 0$
  — for $(i, j) \in A_k^+$, $\quad W_k^+ = W_k^+ + S_{i,j} e^{-M_{i,j}}$
  — for $(i, j) \in A_k^-$, $\quad W_k^- = W_k^- + S_{i,j} e^{-M_{i,j}}$
  — $G_k = \left| \sqrt{W_k^+} - \sqrt{W_k^-} \right|$

- Choose $k^* = \arg\max_k G_k$, and $\delta^* = \frac{1}{2} \log \frac{W_{k^*}^+ + \epsilon Z}{W_{k^*}^- + \epsilon Z}$
- for $(i, j) \in A_{k^*}^+$, $\quad M_{i,j} = M_{i,j} + \delta^*$
- for $(i, j) \in A_{k^*}^-$, $\quad M_{i,j} = M_{i,j} - \delta^*$
- $\bar{\alpha}^t = \text{Upd}(\bar{\alpha}^{t-1}, k^*, \delta^*)$

Output Final parameter setting $\bar{\alpha}^N$

Figure 3
A naive algorithm for the boosting loss function.

4.5 A New, More Efficient, Algorithm for ExpLoss

This section presents a new algorithm, which is equivalent to the ExpLoss algorithm in figure 3, but can be vastly more efficient for problems with sparse feature spaces. In the experimental section of this paper we show that it is almost 2,700 times more efficient for our task than the algorithm in figure 3. 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.

The new algorithm is also applicable, with minor modifications, to boosting approaches for classification problems where the representation also involves sparse binary features (for example the text classification problems in (Schapire and Singer 2000)). As far as we are aware, the new algorithm has not appeared elsewhere in the boosting literature.

Figure 4 shows the improved boosting algorithm. Inspection of the algorithm in figure 3 shows that only margins on examples in the sets $A^+_{k^*}$ and $A^-_{k^*}$ are modified when a feature k^* is selected. The feature space in many NLP problems is very sparse (most features only appear on relatively few training examples, or equivalently, most training examples will only have a few non-zero features). It follows that in many cases, the sets $A^+_{k^*}$ and $A^-_{k^*}$ will be much smaller than the overall size of the training set. Therefore when updating the model from $\bar{\alpha}$ to $\mathrm{Upd}(\bar{\alpha}, k^*, \delta^*)$ 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 in figure 4 only recalculates the values of W^+_k and W^-_k for those features which co-occur with the selected feature k^*.

To achieve this, the algorithm relies on a second pair of indices. For all i, $2 \leq j \leq n_i$ we define

$$\begin{aligned} B^+_{i,j} &= \{k : [h_k(x_{i,1}) - h_k(x_{i,j})] = 1\} \\ B^-_{i,j} &= \{k : [h_k(x_{i,1}) - h_k(x_{i,j})] = -1\} \end{aligned} \tag{18}$$

So $B^+_{i,j}$ and $B^+_{i,j}$ are indices from training examples to features. With the algorithm in figure 4, updating the values of W^+_k and W^-_k for the features which co-occur with k^* involves the following number of steps:

$$C = \sum_{(i,j) \in A^+_{k^*}} \left(|B^+_{i,j}| + |B^-_{i,j}|\right) + \sum_{(i,j) \in A^-_{k^*}} \left(|B^+_{i,j}| + |B^-_{i,j}|\right) \tag{19}$$

In contrast, the naive algorithm requires a pass over the entire training set, which requires the following number of steps:

$$T = \sum_{i=1}^{n} \sum_{j=2}^{n_i} \left(|B^+_{i,j}| + |B^-_{i,j}|\right) \tag{20}$$

The relative efficiency of the two algorithms depends on the value of C/T at each iteration. In the worst case, when every feature chosen appears on every training example, then $C/T = 1$, and the two algorithms essentially have the same running time. However in sparse feature spaces there is reason to believe that C/T will be small for most iterations. In section 6.4.3 we show that this is the case for our experiments.

4.6 Feature Selection for LogLoss

We now describe an approach that was implemented for LogLoss. At the first iteration, α_0 is set to 1. Feature selection then searches for values of the remaining parameters,

Input

Examples $x_{i,j}$ for $i = 1 \ldots n$, $j = 1 \ldots n_i$ drawn from some set $\mathcal{X}$. Weights $S_{i,j}$ representing importance of examples. Initial model log-likelihoods $L(x_{i,j})$, for all examples $x_{i,j}$. Feature functions $h_k : \mathcal{X} \rightarrow \{0, 1\}$ for $k = 1 \ldots m$. Smoothing parameter ϵ (usually chosen by optimization on development data). Number of rounds N (usually chosen by optimization on development data).

Initialize

- Set $\alpha_0 = \arg\min_\alpha \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-\alpha[L(x_{i,1})-L(x_{i,j})]}$.
- Set $\alpha_k = 0$ for $k = 1 \ldots m$.
- For all i, $2 \leq j \leq n_i$ set margins $M_{i,j} = \alpha_0 \, [L(x_{i,1}) - L(x_{i,j})]$.
- For all $k = 1 \ldots m$, set

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

- For all i, $2 \leq j \leq n_i$ set

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

- Calculate Z, and W_k^+, W_k^-, $G_k = \left| \sqrt{W_k^+} - \sqrt{W_k^-} \right|$ for $k = 1 \ldots m$ using the algorithm in figure 3.

Repeat for $t = 1$ to N

- Choose $k^* = \arg\max_k G_k$, and $\delta^* = \frac{1}{2} \log \frac{W_{k^*}^+ + \epsilon Z}{W_{k^*}^- + \epsilon Z}$.
- for $(i,j) \in A_{k^*}^+$
  - — set $\Delta = S_{i,j} \left(e^{-M_{i,j} - \delta^*} - e^{-M_{i,j}} \right)$, set $M_{i,j} = M_{i,j} + \delta^*$, and $Z = Z + \Delta$
  - — for $k \in B_{i,j}^+$, $\quad W_k^+ = W_k^+ + \Delta$
  - — for $k \in B_{i,j}^-$, $\quad W_k^- = W_k^- + \Delta$
- for $(i,j) \in A_{k^*}^-$
  - — set $\Delta = S_{i,j} \left(e^{-M_{i,j} + \delta^*} - e^{-M_{i,j}} \right)$, set $M_{i,j} = M_{i,j} - \delta^*$, and $Z = Z + \Delta$
  - — for $k \in B_{i,j}^+$, $\quad W_k^+ = W_k^+ + \Delta$
  - — for $k \in B_{i,j}^-$, $\quad W_k^- = W_k^- + \Delta$
- For features k whose values of W_k^+ and/or W_k^- have changed, update G_k.
- $\bar{\alpha}^t = \mathrm{Upd}(\bar{\alpha}^{t-1}, k^*, \delta^*)$

Output Final parameter setting $\bar{\alpha}^N$

Figure 4
An improved algorithm for the boosting loss function.

$\alpha_1 \ldots \alpha_m$. 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 a particular parse $x_{i,q}$ given parameter settings $\bar{\alpha}$:

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

Recall that the log-loss is

$$\text{LogLoss}(\bar{\alpha}) = \sum_i -\log P(x_{i,1} \mid s_i, \bar{\alpha})$$

Unfortunately, unlike the case of ExpLoss, in general an analytic solution for $BestWt$ 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:

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

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 $\text{LogLoss}(\text{Upd}(\bar{\alpha}, k, \tilde{\delta})) \leq \text{LogLoss}(\bar{\alpha})$), with equality holding only when α_k is already at the optimal value, in other words when $\arg\min_\delta \text{LogLoss}(\text{Upd}(\bar{\alpha}, k, \delta)) = 0$. 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 δ:

- **a** Calculate $\tilde{p}_k(\bar{\alpha}')$
- **b** $\delta \leftarrow \delta + \log \frac{\tilde{h}_k}{\tilde{p}_k(\bar{\alpha}')}$
- **c** $\bar{\alpha}' \leftarrow \text{Upd}(\bar{\alpha}, k, \delta)$

3 Return $BestWt(k, \bar{\alpha}) = \delta$

Given this method for calculating $BestWt(k, \bar{\alpha})$, $BestLoss(k, \bar{\alpha})$ can be calculated as $Loss(k, BestWt(k, \bar{\alpha}))$. Note that this is only one of a number of methods for finding $BestWt(k, \bar{\alpha})$: given that this is a one parameter, convex optimization problem, it is fairly simple task and there are many methods which could be used.

Unfortunately there does not appear to be an efficient algorithm for LogLoss that is analogous to the ExpLoss algorithm in figure 4 (at least if the feature selection method is required to pick the feature with highest impact on the loss function at each iteration). A similar observation for LogLoss 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. However this observation does not lead to an efficient algorithm – updating these values is much more expensive than in the ExpLoss case. The procedure for finding the optimal value $BestWt(k, \bar{\alpha})$ must be applied for each feature which co-occurs with the chosen feature k^*. For example, the iterative scaling procedure described above must be applied for a number of features. For each feature, this will involve recalculation of the distribution $\{P(x_{i,1} \mid s_i), P(x_{i,2} \mid s_i) \ldots P(x_{i,n_i} \mid s_i)\}$ for each example i on which the feature occurs.[7] It only takes one feature to be seen on all training examples for the algorithm to involve recalculation of $P(x_{i,j} \mid s_i)$ for the entire training set. This contrasts with the simple updates in the improved boosting algorithm ($W_k^+ = W_k^+ + \Delta$ and $W_k^- = W_k^- + \Delta$). In fact in the parsing experiments we were forced to give up on the LogLoss feature selection methods due to their inefficiency.

Note, however, that approximate methods for finding the best feature and updating its weight may lead to efficient algorithms. Appendix B gives a sketch of one such approach, which is based on results from (Collins, Schapire and Singer, 2002). We did not test this method – we leave this to future work.

5 Related Work

5.1 History-Based Models with Complex Features

(Charniak 2000) describes a parser which incorporates additional features to a previously developed parser, that of (Charniak 1997). The method gives substantial improvements over the original parser, and results which are very close to the results of the boosting method we have described in this paper (see section 6 for experimental results comparing the two methods). Our features are in many ways similar to those of (Charniak 2000). The model in (Charniak 2000) is quite different, however. The additional features are incorporated using a method inspired by maximum-entropy models (e.g., the model of (Ratnaparkhi 1997)).

(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. As a result the model can take into account a quite rich set of features in the history.

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 Markov Random Field (LogLoss) models have more flexibility in terms of the features that can be included in the model.

5.2 Joint Markov Random Fields

(Abney 1997) describes the application of Markov Random Fields to stochastic HPSG grammars. (Della Pietra et al. 1997) describe feature selection methods for MRFs, and (Rosenfeld 1997) describe 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 (or structures in question): for this reason we describe these approaches as "Joint Markov Random Fields" (JRFs). The probability of a tree $x_{i,j}$ is

$$P(x_{i,j}) = \frac{e^{F(x_{i,j})}}{\sum_{x \in \mathcal{Z}} e^{F(x)}} \tag{21}$$

7 This is not a failure of iterative scaling alone: given that in the general case closed form solutions for $BestWt$ and $BestLoss$ do not exist, it is hard to imagine a method that computes these values exactly without some kind of iterative method which requires repeatedly visiting the examples on which a feature is seen.

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 Monte-Carlo methods are proposed in (Della Pietra et al. 1997; Abney 1997; Rosenfeld 1997) as a technique for estimation of this value. Our sense is that these methods can be computationally expensive. Notice that the joint likelihood in Equation (21) 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.

5.3 Conditional Markov Random Fields

(Ratnaparkhi et al., 1994; Johnson et al. 1999) suggested training the conditional or pseudo-likelihood of a MRF, the LogLoss function in Equation (9). (Ratnaparkhi et al., 1994) use feature selection techniques for the task. (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:

$$\bar{\alpha}^* = \arg\min_{\bar{\alpha}} \left(\text{LogLoss}(\bar{\alpha}) + \sum_{k=0\ldots m} \frac{\alpha_k^2}{\phi_k^2} \right) \tag{22}$$

Closed-form updates under iterative scaling are not possible with this objective function, instead optimization algorithms such as gradient descent or conjugate gradient methods are used to estimate parameter values.

In more recent work, (Lafferty et al., 2001) describe the use of conditional Markov Random Fields (CRFs) for tagging tasks such as named entity recognition or part-of-speech tagging (Hidden Markov Models are a common method applied to these tasks). A key insight of (Lafferty et al., 2001) is that when features are of a significantly local nature, the gradient of the function in Equation (22) can be calculated efficiently using dynamic programming, even in cases where the set of candidates involves all possible tagged sequences, and is therefore exponential in size. See also (Sha and Pereira 2003) for more recent work on CRFs.

5.4 Boosting, Perceptron and Support Vector Machine Approaches for Ranking Problems

(Freund et al. 1998) introduced a formulation of boosting for ranking problems. The problem we have considered is a special case of the problem in (Freund et al. 1998), in that we have considered a binary distinction between candidates (i.e., the best parse vs. other parses), whereas (Freund et al. 1998) consider learning full or partial orderings over candidates. The improved algorithm that we have introduced in figure 4 is however, a new algorithm, which could perhaps be generalized to the full problem of (Freund et al. 1998) – we leave this to future research.

(Altun, Hofmann, and Johnson 2003) describe experiments on tagging tasks using the ExpLoss function, in contrast to the LogLoss function used in (Lafferty et al., 2001). (Altun, Hofmann, and Johnson 2003) describe how dynamic programming methods can be used to calculate gradients of the ExpLoss function even in cases where the set of candidates again includes all possible tagged sequences, a set which grows exponentially in size with the length of the sentence being tagged.

Perceptron-based algorithms, or the voted perceptron approach of (Freund and Schapire, 1999), are another alternative to boosting and LogLoss methods. See (Collins, 2002a; Collins, 2002b; Collins and Duffy, 2002; Collins and Duffy, 2001) for applications of the perceptron algorithm. (Collins, 2002b) gives convergence proofs for the methods; (Collins, 2002a) directly compares the boosting and perceptron approaches on a named entity task; and (Collins and Duffy, 2002; Collins and Duffy, 2001) use a reranking approach with kernels, which allow representations of parse trees or labelled sequences in very high dimensional spaces.

(Shen, Sarkar and Joshi 2003) describes support vector machine approaches to ranking problems, and applies Support Vector Machines (SVMs) using Tree Adjoining Grammar (Joshi, Levy and Takahashi 1975) features to the parsing datasets we have described in this paper, with good empirical results.

See (Collins to appear) for a discussion of many of these methods, including an overview of statistical bounds for the boosting, perceptron and SVM methods, as well as a discussion of the computational issues of the different algorithms.

6 Experimental Evaluation

6.1 Generation of Parsing Datasets

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 the main training set. 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. In order to gain a representative set of training data, the 36,000 training sentences were parsed in 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.

In the experiments we used the following definition for the *Score* of the parse:

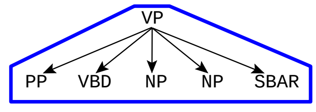
$$Score(x_{i,j}) = \frac{\text{F-measure}(x_{i,j})}{100} \times \text{Size}(x_{i,j})$$

where F-measure($x_{i,j}$) is the F_1 score[8] of the parse when compared to the gold-standard parse (a value between 0 and 100), and Size($x_{i,j}$) is the number of constituents in the gold standard parse for the i'th sentence. Hence the *Score* function is sensitive to both the accuracy of the parse, and also the number of constituents in the gold-standard parse.

6.2 Features

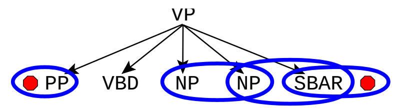
The following types of features were included in the model. We will use the rule `VP -> PP VBD NP NP SBAR` with head `VBD` as an example. Note that the output of our baseline parser produces syntactic trees with headword annotations.

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

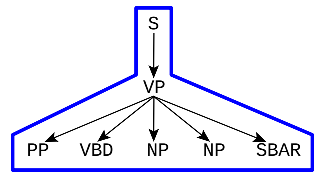


8 Note that in the rare cases that the baseline parser produces no constituents, the precision is undefined – in these cases we defined the F-measure to be 0.

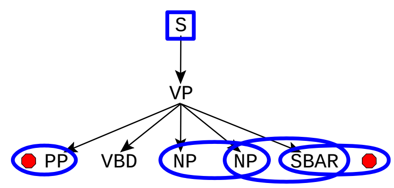
Bigrams These are adjacent pairs of non-terminals to the left and right of the head. As shown, the example rule would contribute the bigrams `(Right,VP,NP,NP)`, `(Right,VP,NP,SBAR)`, `(Right,VP,SBAR,STOP)`, and `(Left,VP,PP,STOP)` to the left of the head.



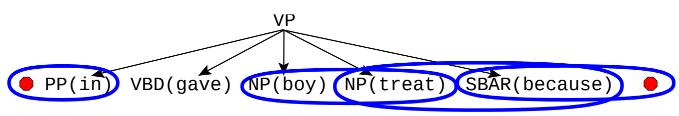
Grandparent Rules Same as **Rules**, but also including the non-terminal above the rule.



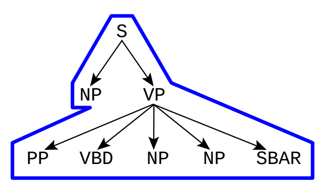
Grandparent Bigrams Same as **Bigrams**, but also including the non-terminal above the bigrams.



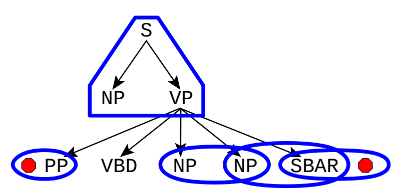
Lexical Bigrams Same as **Bigrams**, but with the lexical heads of the two non-terminals also included.



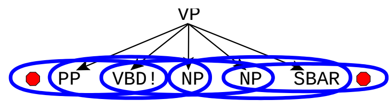
Two-level Rules Same as **Rules**, but also including the entire rule above the rule.



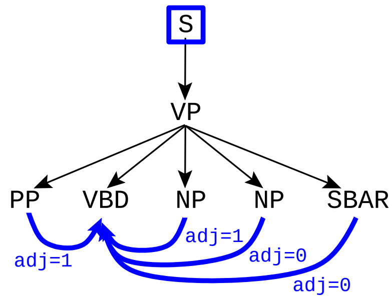
Two-level Bigrams Same as **Bigrams**, but also including the entire rule above the rule.



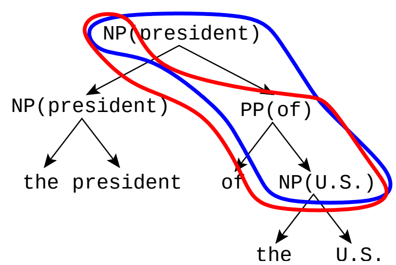
Trigrams All trigrams within the rule. The example rule would contribute the trigrams (VP,STOP,PP,VBD!), (VP,PP,VBD!,NP), (VP, VBD!,NP,NP), (VP,NP,NP,SBAR) and (VP,NP, SBAR,STOP) (! is used to mark the head of the rule).



Head-Modifiers All head-modifier pairs, with the grandparent non-terminal also included. An adj flag is also included, which is 1 if the modifier is adjacent to the head, 0 otherwise. As an example, say the non-terminal dominating the example rule is S. The example rule would contribute (Left,S,VP,VBD,PP,adj=1), (Right,S,VP,VBD,NP,adj=1), (Right,S,VP, VBD,NP,adj=0), and (Right,S,VP,VBD,SBAR, adj=0).



PPs Lexical trigrams involving the heads of arguments of prepositional phrases. The example shown at right would contribute the trigram (NP,NP,PP,NP,president,of,U.S.), in addition to the more general trigram 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,$= dist$), and (VP,VBD,SBAR,$\leq x$) for all $dist \leq x \leq 9$ and (VP,VBD,SBAR,$\geq x$) 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).

6.3 Applying the Reranking Methods

The ExpLoss method was trained with several values for the smoothing parameter ϵ: $\{0.0001, 0.0025, 0.005, 0.0075, 0.01, 0.025, 0.05, 0.075\}$. For each value of ϵ the method was run for $100,000$ rounds on the training data. The implementation was such that the feature updates for all $100,000$ rounds for each training run were recorded in a file. This made it simple to test the model on development data for all values of N between 0 and $100,000$.

The different values of ϵ and N were compared on development data through the following criterion:

$$\sum_i Score(z_i) \tag{23}$$

where $Score$ is as defined above, and z_i is the output of the model on the i'th development set example. The ϵ, N values which maximized this quantity were used to define the final model applied to the test data (section 23 of the treebank). The optimal values were $\epsilon = 0.025$ and $N = 90,386$, at which point 11,673 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 3-4 hours on a 1.6 GHz pentium processor machine, and around 2 Gigabytes of memory.

Table 1 shows results for the method. (Collins 1999) was the base model; the ExpLoss model gave a 1.5% absolute improvement over this method. The method gives very similar accuracy to the model of (Charniak 2000), which also uses a rich set of initial features in addition to Charniak's original model (Charniak 1997).

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.[9] On an older machine (an order of magnitude or more slower than the machine used for the final tests) 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). This was in spite of various heuristics that were implemented in an attempt to speed up LogLoss – for example, selecting multiple features at each round, or recalculating the statistics for only the best K features for some small K at the previous round of feature selection. In initial experiments we found ExpLoss to give similar, perhaps slightly better, accuracy than LogLoss.

6.4 Further Experiments

This section describes further experiments investigating various aspects of the boosting algorithm: the effect of the ϵ and N parameters, learning curves, the choice of the $S_{i,j}$ weights, and efficiency issues.

6.4.1 The Effect of the ϵ and N Parameters Figure 5 shows the learning curve on development data for the optimal value of ϵ (i.e., $\epsilon = 0.0025$). The accuracy shown is the relative performance to the baseline method of using the probability from the generative model alone in ranking parses, where the measure in Equation (23) is used to measure performance. For example, a score of 101.5 indicates a 1.5% increase in this score. The learning curve is initially steep, eventually flattening off, but reaching its peak value after a large number (90,386) of rounds of feature selection.

Table 2 indicates how the peak performance varies with the smoothing parameter ϵ. Figure 6 shows learning curves for various values of ϵ. It can be seen that values other than $\epsilon = 0.0025$ can lead to undertraining or overtraining of the model.

6.4.2 The Effect of the $S_{i,j}$ Weights on Examples In section 4.2.3 we introduced the idea of weights $S_{i,j}$ representing the importance of examples. Thus far, in the experiments

9 All features described above except **Distance Head-Modifiers** and **Further Lexicalization** were included.

MODEL	≤ 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%
CH00	90.1%	90.1%	0.74	70.1%	89.6%
ExpLoss	90.2%	90.4%	0.73	71.2%	90.2%
MODEL	≤ 100 Words (2416 sentences)				
	LR	LP	CBs	0 CBs	2 CBs
CH97	86.7%	86.6%	1.20	59.5%	83.2%
RA98	86.3%	87.5%	1.21	60.2%	—
CO99	88.1%	88.3%	1.06	64.0%	85.1%
CH00	89.6%	89.5%	0.88	67.6%	87.7%
ExpLoss	89.6%	89.9%	0.86	68.7%	88.3%

Table 1
Results on Section 23 of the WSJ Treebank. **LR/LP** = labelled 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. All the results in this table are for models trained and tested on the same data, using the same evaluation metric. CH97 = Charniak 1997, RA98 = Ratnaparkhi 1998, CH00 = Charniak 2000, CO99 = Collins 1999. Note that the ExpLoss results are very slightly different from the original results published in (Collins, 2000). We recently reimplemented the boosting code, and re-ran the experiments, and minor differences in the code and ϵ values tested on development data lead to minor improvements in the results.

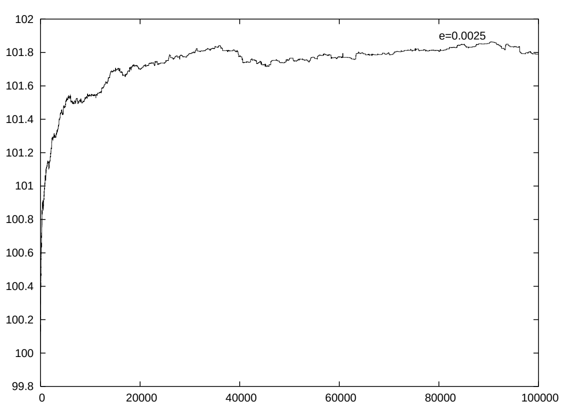


Figure 5
Graphs showing the learning curve on development data for the optimal value for ϵ, i.e., $\epsilon = 0.0025$. The y-axis is the level of accuracy (100 is the baseline score), and the x-axis is the number of rounds of boosting.

in this paper we have used the definition

$$S_{i,j} = Score(x_{i,1}) - Score(x_{i,j}) \tag{24}$$

thereby weighting examples in proportion to their difference in score from the correct parse for the sentence in question. In this section we compare this approach to a default definition of $S_{i,j}$, namely

$$S_{i,j} = 1 \tag{25}$$

Using this definition, we trained the ExpLoss method on the same training set for

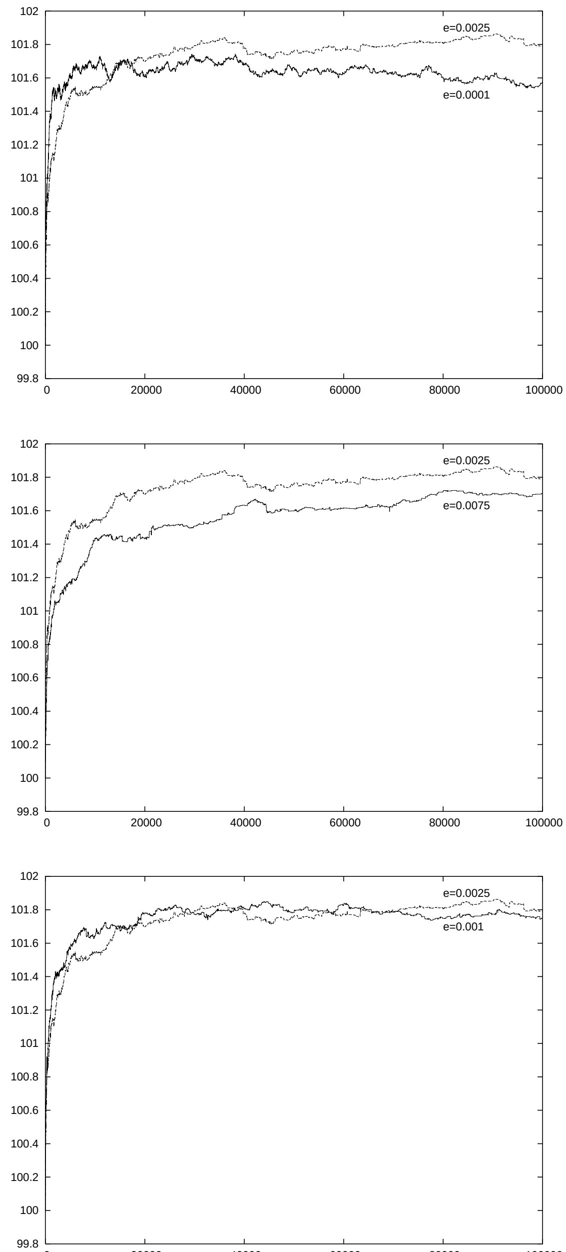


Figure 6
Graphs showing learning curves on development data for various values of ϵ. In each case the y-axis is the level of accuracy (100 is the baseline score), and the x-axis is the number of rounds of boosting. The three graphs compare the curve for $\epsilon = 0.0025$ (the optimal value) to $\epsilon = 0.0001$, $\epsilon = 0.0075$, and $\epsilon = 0.001$ respectively. The first graph shows that $\epsilon = 0.0001$ leads to under-smoothing (overtraining). Initially the graph is higher than that for $\epsilon = 0.0025$, but on later rounds the performance starts to decrease. The second graph shows that $\epsilon = 0.0075$ leads to over-smoothing (undertraining). The graph shows consistently lower performance than that for $\epsilon = 0.0025$. The third graph shows that there is little difference in performance for $\epsilon = 0.001$ vs. $\epsilon = 0.0025$.

ϵ	Best N	Best Score
0.0001	29,471	101.743
0.00025	22,468	101.849
0.0005	48,795	101.845
0.00075	43,386	101.809
0.001	43,975	101.849
0.0025	90,386	101.864
0.005	66,378	101.824
0.0075	80,746	101.722

Table 2
Table showing peak performance achieved for various values of ϵ. "Best N" refers to the number of rounds at which peak development set accuracy was reached. "Best Score" indicates the relative performance compared to the baseline method, at the optimal value for N.

ϵ	Best Score (weighted)	Best Score (unweighted)
0.0001	101.743	101.744
0.00025	101.849	101.754
0.0005	101.845	101.778
0.00075	101.809	101.762
0.001	101.849	101.778
0.0025	101.864	101.699
0.005	101.824	101.61
0.0075	101.722	101.604

Table 3
Table showing peak performance achieved for various values of ϵ for $S_{i,j} = Score(x_{i,1}) - Score(x_{i,j})$ (column labelled "weighted") and $S_{i,j} = 1$ (column labelled "unweighted").

several values of the smoothing parameter ϵ, and evaluated the performance on development data. Table 3 compares the peak performance achieved under the two definitions of $S_{i,j}$ on the development set. It can be seen that the definition in Equation (24) outperforms the simpler method in Equation (25). Figure 7 shows the learning curves for the optimal values of ϵ for the two methods. It can be seen that the learning curve for the definition of $S_{i,j}$ in Equation (24) consistently dominates the curve for the simpler definition.

6.4.3 Efficiency Gains Section 4.5 introduced an efficient algorithm for optimizing ExpLoss. In this section we explore the empirical gains in efficiency seen on the parsing datasets in this paper.

We first define the quantity T as follows:

$$T = \sum_i \sum_{j=2}^{n_i} \left(|B_{i,j}^+| + |B_{i,j}^-| \right)$$

This is a measure of the number of updates to the W_k^+ and W_k^- variables required in making a pass over the entire training set. Thus this is a measure of the amount of computation that the naive algorithm for ExpLoss, in figure 3, requires for each round of feature selection.

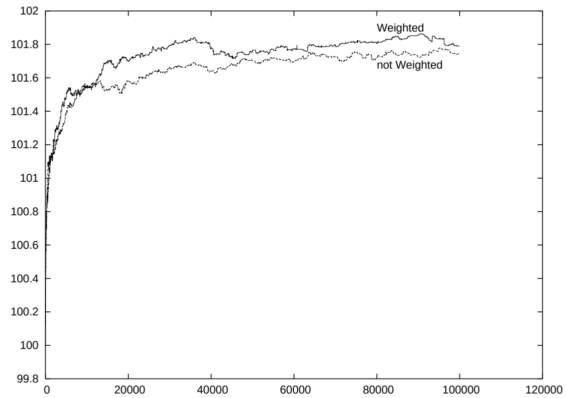


Figure 7
Figure showing performance vs. number of rounds of boosting for $S_{i,j} = Score(x_{i,1}) - Score(x_{i,j})$ (curve labelled "weighted") and $S_{i,j} = 1$ (curve labelled "not weighted").

Next, say the improved algorithm in figure 4 selects feature k^* on the t'th round of feature selection. Then we define the following quantity:

$$C_t = \sum_{(i,j) \in A^+_{k*}} \left(|B^+_{i,j}| + |B^-_{i,j}|\right) + \sum_{(i,j) \in A^-_{k*}} \left(|B^+_{i,j}| + |B^-_{i,j}|\right)$$

This is a measure of the number of summations required by the improved algorithm in figure 4 at the t'th round of feature selection.

We are now in a position to compare the running times of the two algorithms. We define the following quantities:

$$Work(n) = \sum_{t=1}^{n} \frac{C_t}{T} \quad (26)$$

$$Savings(n) = \frac{nT}{\sum_{t=1}^{n} C_t} \quad (27)$$

$$Savings(a, b) = \frac{(1 + b - a)T}{\sum_{t=a}^{b} C_t} \quad (28)$$

Here, $Work(n)$ is the computation required for n rounds of feature selection, where a single unit of computation corresponds to a pass over the entire training set. $Savings(n)$ tracks the relative efficiency of the two algorithms as a function of the number of features, n. For example, if $Savings(100) = 1,200$ this signifies that for the first 100 rounds of feature selection the improved algorithm is $1,200$ times as efficient as the naive algorithm. Finally, $Savings(a, b)$ indicates the relative efficiency between rounds a and b of feature selection inclusive. For example, $Savings(11, 100) = 83$ signifies that between rounds 11 and 100 inclusive of the algorithm, the improved algorithm was 83 times as efficient.

Figures 8 and 9 show graphs of $Work(n)$ and $Savings(n)$ vs. n. The savings from the improved algorithm are dramatic. In $100,000$ rounds of feature selection the improved algorithm requires total computation that is equivalent to a mere 37.1 passes over the training set. This is a saving of a factor of 2692 over the naive algorithm.

Table 4 shows the value of $Savings(a, b)$ for various values of (a, b). It can be seen that the performance gains are significantly larger in later rounds of feature selection,

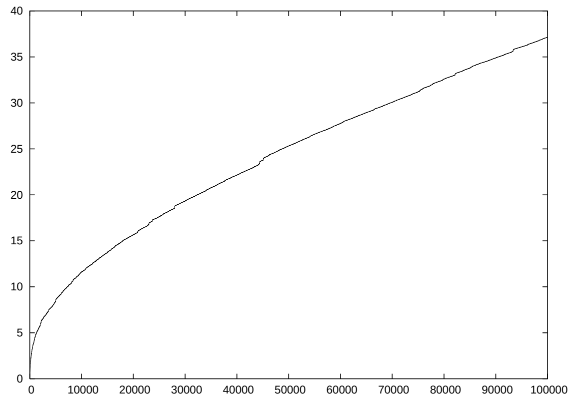


Figure 8
A graph of $Work(n)$ (y-axis) vs. n (x-axis).

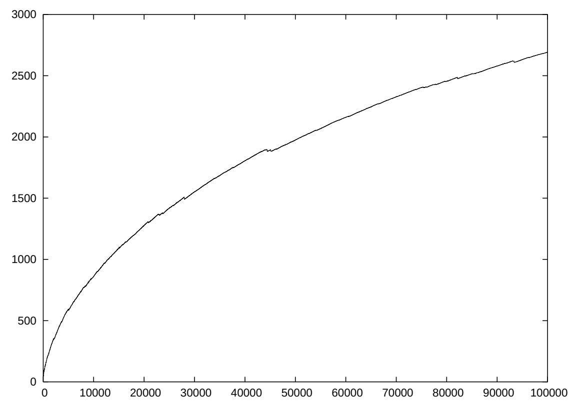


Figure 9
A graph of $Savings(n)$ (y-axis) vs. n (x-axis).

a-b	$Savings(a, b)$
1-100,000	2692.7
1-10	48.6
11-100	83.5
101-1,000	280.0
1001-10,000	1263.9
10001-50,000	2920.2
50001-100,000	4229.8

Table 4
Values of $Savings(a, b)$ for various values of a, b.

presumably because in later stages relatively infrequent features are being selected. Even so, there are still savings of a factor of almost 50 in the early stages of the method.

7 Conclusions

This paper has introduced a new algorithm, based on boosting approaches in machine learning, to ranking problems in natural language processing. The approach gives a 13% relative reduction in error on parsing Wall Street Journal data. While in this paper the experimental focus is on parsing, many other problems in natural language processing or speech recognition can also be framed as reranking problems, so the methods described should be quite broadly applicable. The boosting approach to ranking has been applied to named-entity segmentation (Collins, 2002a) and natural language generation (Walker et al., 2001). The key characteristics of the approach are the use of global features, and the use of a training criterion (optimization problem) that is discriminative, and closely related to the task in hand – i.e., parse accuracy.

In addition, the paper introduced a new algorithm for the boosting approach which takes advantage of the sparse nature of the feature space in the parsing data that we use. Other NLP tasks are likely to have similar characteristics in terms of sparsity. Experiments show an efficiency gain of a factor of over 2,600 on the parsing data for the new algorithm over the obvious implementation of the boosting approach. We would argue that the improved boosting algorithm is a natural alternative to maximum-entropy or (Conditional) Markov Random Field methods. The paper has drawn connections between boosting and maximum-entropy models in terms of the optimization problems that they involve, the algorithms used, their relative efficiency, and their performance in empirical tests.

References

Abney, Steven. (1997). Stochastic Attribute-Value Grammars. *Computational Linguistics,* 23(4):597-618.

Altun, Yasemin, Thomas Hofmann and Mark Johnson. (2003). Discriminative Learning for Label Sequences via Boosting. In *Advances in Neural Information Processing Systems (NIPS 15).*

Berger, Adam L. and Stephen A. Della Pietra,and Vincent J. Della Pietra. (1996). A Maximum Entropy Approach to Natural Language Processing. *Computational Linguistics*, 22(1):39-71.

Black, Ezra, Frederick Jelinek, John Lafferty, David Magerman, Robert Mercer and Salim Roukos. (1992). Towards History-Based Grammars: Using Richer Models for Probabilistic Parsing. In *Proceedings of the 5th DARPA Speech and Natural Language Workshop*, Harriman, NY.

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

Charniak, Eugene. (2000). A Maximum-Entropy-Inspired Parser. In *Proceedings of NAACL-2000.*

Collins, Michael. (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*, pages 16-23.

Collins, Michael. (1999). Head-Driven Statistical Models for Natural Language Parsing. PhD Dissertation, University of Pennsylvania.

Collins, Michael. (2000). Discriminative Reranking for Natural Language Parsing. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000).* San Francisco: Morgan Kaufmann.

Collins, Michael, and Nigel Duffy. (2001). Convolution Kernels for Natural Language. In *Advances in Neural Information Processing Systems (NIPS 14).*

Collins, Michael, Robert E. Schapire, and Yoram Singer. (2002). Logistic regression, AdaBoost and Bregman distances. *Machine Learning*, 48(1/2/3).

Collins, Michael, and Nigel Duffy. (2002). New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron. In *Proceedings of ACL 2002*.

Collins, Michael. (2002a). Ranking Algorithms for Named–Entity Extraction: Boosting and the Voted Perceptron. In *Proceedings of ACL 2002*.

Collins, Michael. (2002b). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with the Perceptron Algorithm. In *Proceedings of EMNLP 2002*.

Collins, Michael. To appear. Parameter Estimation for Statistical Parsing Models: Theory and Practice of Distribution-Free Methods. To appear as a book chapter.

Della Pietra, Stephen, Vincent Della Pietra and John Lafferty. (1997). Inducing Features of Random Fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 19(4), April 1997, pp. 380-393.

Duffy, Nigel and David Helmbold. (1999). Potential Boosters? In *Advances in Neural Information Processing Systems (NIPS 12)*.

Paper on loss functions of margins.

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

Freund, Yoav, Raj Iyer, Robert E. Schapire and Yoram Singer. (1998). An efficient boosting algorithm for combining preferences. Extended abstract appeared in Machine Learning: Proceedings of the Fifteenth International Conference, 1998.

Freund, Yoav and Robert E. Schapire. (1999). Large Margin Classification using the Perceptron Algorithm. In *Machine Learning*, 37(3):277–296.

Friedman, Jerome H., Trevor Hastie, and Robert Tibshirani. (1998). Additive Logistic Regression: a Statistical View of Boosting.

Hoffgen, Klauss U., Kevin S. van Horn, and Hans U. Simon. (1995). Robust trainability of single neurons. Journal of Computer and System Sciences, 50:114–125, 1995.

Johnson, Mark, Stuart Geman, S. Canon, Z. Chi and S. Riezler (1999). Estimators for Stochastic "Unification-based" Grammars. Proceedings of the ACL 1999.

Joshi, Aravind K., L. Levy and M. Takahashi. (1975). Tree adjunct grammar. In *Journal of Computer and System Science*, vol. 21, no. 2, 1975.

Lafferty, John. (1999). Additive models, boosting, and inference for generalized divergences. In Proceedings of the Twelfth Annual Conference on Computational Learning Theory (COLT'99), 1999.

Lafferty, John, Andrew McCallum, and Fernando Pereira. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of ICML 2001.*

Lebanon, Guy and John Lafferty. (2002). Boosting and maximum likelihood for exponential models. In *Advances in Neural Information Processing Systems (NIPS 14).*

Malouf, Robert. (2002). A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of the Sixth Conference on Natural Language Learning (CoNNL-2002).*

Marcus, Mitchell, Beatrice Santorini and M. Marcinkiewicz. (1993). Building a Large Annotated Corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313-330.

Mason, Llew, Peter L. Bartlett and Jonathan Baxter. (1999). Direct Optimization of Margins Improves Generalization in Combined Classifiers. In *Advances in Neural Information Processing Systems (NIPS 12).*

Och, Franz Josef, and Hermann Ney. (2002). Discriminative Training and Maximum Entropy Models for Statistical Machine Translation. In *ACL 2002: Proc. of the 40th Annual Meeting of the Association for Computational Linguistics* pp. 295-302, Philadelphia, PA.

Papineni, Kishore A., Salim Roukos, and R. T. Ward. (1997). Feature-based language understanding. In *Proceedings of EuroSpeech'97*, Rhodes, Greece, vol 3, pp. 1435-1438.

Papineni, Kishore A., Salim Roukos, and R. T. Ward. (1998). Maximum likelihood and discriminative training of direct translation models. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing*, Seattle, vol 1, pp. 189-192.

Pearl, Judea. (1988). Probabilistic Reasoning in Intelligent Systems, Morgan Kaufman, San Mateo, Ca.

Ratnaparkhi, Adwait, Salim Roukos, and R. T. Ward. (1994). A Maximum Entropy Model for Parsing. In *Proceedings of the International Conference on Spoken Language Processing*, pages 803-806. Yokohama, Japan.

Ratnaparkhi, Adwait. (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*, Brown University, Providence, Rhode Island.

Ratnaparkhi, Adwait. (1998). Maximum Entropy Models for Natural Language Ambiguity Resolution. Ph. D thesis, University of Pennsylvania.

R. Rosenfeld. (1997). A Whole Sentence Maximum Entropy Language Model, Proc. IEEE workshop on Speech Recognition and Understanding, Santa Barbara, California, December 1997.

Schapire Robert E., Yoav Freund, Peter Bartlett and W. S. Lee. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 26(5):1651-1686.

Schapire, Robert E. and Yoram Singer. (1998). Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pages 80–91, 1998. To appear, *Machine Learning*.

Schapire, Robert E. and Yoram Singer. (2000). BoosTexter: A boosting-based system for text categorization. Machine Learning, 39(2/3):135-168, 2000.

Sha, Fei and Fernando Pereira. (2003). Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003.*

Shen, Libin, Anoop Sarkar and Aravind K. Joshi. (2003). Using LTAG Based Features in Parse Reranking. To appear in *Proceedings of the 2003 conference on Empirical Methods in Natural Language Processing (EMNLP 2003).*

Valiant, Leslie G. (1984). A theory of the learnable. In *Communications of the ACM*, 27(11):1134–1142.

Walker, Marilyn, Owen Rambow, and Monica Rogati. (2001). SPoT: a trainable sentence planner. In *Proceedings of the 2nd Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL 2001).*

Appendix

Appendix A: Derivation of Updates for ExpLoss

This appendix gives a derivation of the optimal updates for ExpLoss. The derivation is very close to that in (Schapire and Singer 1998). Recall that for parameter values $\bar{\alpha}$, we need to compute $BestWt(k, \bar{\alpha})$ and $BestLoss(k, \bar{\alpha})$ for $k = 1 \ldots m$, where

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

and

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

The first thing to note is that an update in parameters from $\bar{\alpha}$ to $\text{Upd}(\bar{\alpha}, k, \delta))$ results in 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 (i, j) 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 ExpLoss function can then be written as

$$\begin{aligned}
\text{ExpLoss}(\text{Upd}(\bar{\alpha}, k, \delta)) &= \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-M_{i,j}(\text{Upd}(\bar{\alpha}, k, \delta))} \\
&= \sum_i \sum_{j=2}^{n_i} S_{i,j} e^{-M_{i,j}(\bar{\alpha}) - \delta[h_k(x_{i,1}) - h_k(x_{i,j})]}
\end{aligned}$$

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

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

Given these definitions, we define W_k^+, W_k^- and W_k^0 as

$$\begin{aligned}
W_k^+ &= \sum_{(i,j) \in A_k^+} S_{i,j} e^{-M_{i,j}(\bar{\alpha})} \\
W_k^- &= \sum_{(i,j) \in A_k^-} S_{i,j} e^{-M_{i,j}(\bar{\alpha})} \\
W_k^0 &= \sum_{(i,j) \in A_k^0} S_{i,j} e^{-M_{i,j}(\bar{\alpha})}
\end{aligned}$$

ExpLoss is now rewritten in terms of these quantities:

$$\begin{aligned}\text{ExpLoss}(\text{Upd}(\bar{\alpha},k,\delta)) &= \sum_{(i,j)\in A_k^+} S_{i,j}e^{-M_{i,j}(\bar{\alpha})-\delta} + \sum_{(i,j)\in A_k^-} S_{i,j}e^{-M_{i,j}(\bar{\alpha})+\delta} + \sum_{(i,j)\in A_k^0} S_{i,j}e^{-M_{i,j}(\bar{\alpha})} \\ &= e^{-\delta}W_k^+ + e^{\delta}W_k^- + W_k^0 \qquad \text{(A-1)}\end{aligned}$$

To find the value of δ that minimizes this loss, we set the derivative of (A-1) 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 (A-1) gives the best loss:

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

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

Appendix B: An Alternative Method for LogLoss

In this section we sketch an alternative approach for feature selection in LogLoss, which is potentially an efficient method, at the cost of introducing an approximation in the feature selection method. Until now, we have defined $BestLoss(k,\bar{\alpha})$ to be the minimum of the loss given that the k'th feature is updated an optimal amount:

$$BestLoss(k,\bar{\alpha}) = \min_{\delta} \text{LogLoss}\left(\text{Upd}(\bar{\alpha},k,\delta)\right)$$

In this section we sketch a different approach, based on results from (Collins, Schapire and Singer, 2002), which leads to a very similar algorithm to that for ExpLoss in figures 3 or 4. Take the following definitions (note the similarity to the definitions in Equations 13, 14, 15, 16, with only the definitions for W_k^+ and W_k^- being altered):

$$W_k^+ = \sum_{(i,j)\in A_k^+} q_{i,j} \qquad W_k^- = \sum_{(i,j)\in A_k^-} q_{i,j} \qquad \text{where } q_{i,j} = \frac{e^{-M_{i,j}(\bar{\alpha})}}{1+\sum_{q=2}^{n_i} e^{-M_{i,q}(\bar{\alpha})}} \qquad \text{(B-3)}$$

$$\begin{aligned}BestWt(k,\bar{\alpha}) &= \frac{1}{2}\log\frac{W_k^+}{W_k^-} \qquad \text{(B-4)} \\ BestLoss(k,\bar{\alpha}) &= \text{LogLoss}(\bar{\alpha}) - \left(\sqrt{W_k^+} - \sqrt{W_k^-}\right)^2 \qquad \text{(B-5)}\end{aligned}$$

Note that the ExpLoss computations can be recovered by replacing $q_{i,j}$ in Equation (B-3) by $q_{i,j} = e^{-M_{i,j}(\bar{\alpha})}$. This is essentially the only difference between the new algorithm and the ExpLoss method.

Results from (Collins, Schapire and Singer, 2002) show that under these definitions the following guarantee holds:

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

So it can be seen that the update from $\bar{\alpha}$ to $\text{Upd}(\bar{\alpha}, k, BestWt(k, \bar{\alpha}))$ is guaranteed to decrease LogLoss by at least $\left(\sqrt{W_k^+} - \sqrt{W_k^-}\right)^2$. From these results, the algorithms in figures 3 and 4 could be altered to take the revised definitions of W_k^+ and W_k^- into account. Selecting the feature with the minimum value of $BestLoss(k, \bar{\alpha})$ at each iteration leads to largest guaranteed decrease in LogLoss. Note that this is now an approximation, in that $BestLoss(k, \bar{\alpha})$ is an upper bound on the log-likelihood which may or may not be tight. There are convergence guarantees for the method, however, in that as the number of rounds of feature selection goes to infinity, the LogLoss approaches its minimum value.

The algorithms in figures 3 and 4 could be modified to take the alternative definitions of W_k^+ and W_k^- into account, thereby being modified to optimize LogLoss instead of ExpLoss. The denominator terms in the $q_{i,j}$ definitions in Equation (B-3) may complicate the algorithms somewhat, but it should still be possible to derive relatively efficient algorithms using the technique.

For a full derivation of the modified updates, and quite technical convergence proofs, see (Collins, Schapire and Singer, 2002). We give a sketch of the argument here. First, we show that

$$\text{LogLoss}\left(\text{Upd}(\bar{\alpha}, k, \delta)\right) \leq \text{LogLoss}\left(\bar{\alpha}\right) - W_k^+ - W_k^- + W_k^+ e^{-\delta} + W_k^- e^{\delta} \tag{B-6}$$

This can be derived as follows (in this derivation we use $g_k(x_{i,j}) = h_k(x_{i,1}) - h_k(x_{i,j})$):

$$\begin{aligned}
\text{LogLoss}\left(\text{Upd}(\bar{\alpha}, k, \delta)\right) &= \text{LogLoss}\left(\bar{\alpha}\right) + \text{LogLoss}\left(\text{Upd}(\bar{\alpha}, k, \delta)\right) - \text{LogLoss}\left(\bar{\alpha}\right) \\
&= \text{LogLoss}\left(\bar{\alpha}\right) + \sum_i \log\left(\frac{1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha}) - \delta g_k(x_{i,j})}}{1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})}}\right) \\
&= \text{LogLoss}\left(\bar{\alpha}\right) + \\
&\quad \sum_i \log\left(\frac{1}{1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})}} + \frac{\sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha}) - \delta g_k(x_{i,j})}}{1 + \sum_{j=2}^{n_i} e^{-M_{i,j}(\bar{\alpha})}}\right) \\
&= \text{LogLoss}\left(\bar{\alpha}\right) + \sum_i \log\left(1 - \sum_{j=2}^{n_i} q_{i,j} + \sum_{j=2}^{n_i} q_{i,j} e^{-\delta g_k(x_{i,j})}\right) \qquad \text{(B-7)} \\
&\leq \text{LogLoss}\left(\bar{\alpha}\right) - \sum_i \sum_{j=2}^{n_i} q_{i,j} + \sum_i \sum_{j=2}^{n_i} q_{i,j} e^{-\delta g_k(x_{i,j})} \qquad \text{(B-8)} \\
&= \text{LogLoss}\left(\bar{\alpha}\right) - (W_k^0 + W_k^+ + W_k^-) + W_k^0 + W_k^+ e^{-\delta} + W_k^- e^{\delta} \\
&= \text{LogLoss}\left(\bar{\alpha}\right) - W_k^+ - W_k^- + W_k^+ e^{-\delta} + W_k^- e^{\delta}
\end{aligned}$$

Equation (B-8) can be derived from Equation (B-7) through the bound $\log(1 + x) \leq x$ for all x.

The second step is to minimize the right hand side of the bound in Equation (B-6) with respect to δ. It can be verified that the minimum is found at

$$\delta = \frac{1}{2} \log \frac{W_k^+}{W_k^-}$$

at which value the right-hand-side of Equation (B-6) is equal to

$$\text{LogLoss}(\bar{\alpha}) - \left(\sqrt{W_k^+} - \sqrt{W_k^-}\right)^2$$