Reranking an N-Gram Supertagger

John Chen[*], Srinivas Bangalore[*], Michael Collins[*], and Owen Rambow[†]
∗AT&T Labs–Research, †University of Pennsylvania
{jchen,srini,mcollins}@research.att.com, rambow@unagi.cis.upenn.edu

1. Introduction

As shown by Srinivas (1997), standard n-gram modeling may be used to perform supertag disambiguation with accuracy that is adequate for partial parsing, but in general not sufficient for full parsing. A serious problem is that n-gram modeling usually considers a very small, fixed context and does not perform well with large tag sets, such as those generated by automatic grammar extraction (Xia, 1999; Chen and Vijay-Shanker, 2000; Chiang, 2000). As an alternative, Chen, Bangalore and Vijay-Shanker (1999) introduce class-based supertagging. An example of class tagging is n-best trigram-based supertagging, which assigns to each word the top n most likely supertags as determined by an n-gram supertagging model. Class-based supertagging can be performed much more accurately than supertagging with only a small increase in ambiguity. In a second phase, the most likely candidate from the class is chosen.

In this paper, we investigate an approach to such a choice based on reranking a set of candidate supertags and their confidence scores. RankBoost (Freund *et al.*, 1998) is the boosting algorithm that we use in order to learn to rerank outputs. It also has been used with good effect in reranking outputs of a statistical parser (Collins, 2000) and ranking sentence plans (Walker, Rambow and Rogati, 2001). RankBoost may learn to correct biases that are inherent in n-gram modeling which lead to systematic errors in supertagging (cf. (van Halteren, 1996)). RankBoost can also use a variety of local and long distance features more easily than n-gram-based approaches (cf. (Chen, Bangalore and Vijay-Shanker, 1999)) because it makes sparse data less of an issue.

The outline of this paper is as follows. First, we develop the background and motivations behind the task of reranking the output of an n-best trigram supertagger. Second, we introduce RankBoost as the approach that we adopt in order to train the reranker. Third, we perform an initial set of experiments where the reranker is trained with different feature subsets. Fourth, we perform an in-depth analysis of several reranking models. Fifth, after pointing out causes that at times render the reranker ineffective, we develop and test some new models that attempt to sidestep these limitations. Lastly, after some significance testing results, we state our conclusions and remark on potential future directions.

2. Background and Motivation

In this section, we motivate the desirability of exploring the use of n-best reranking of supertags. Although we give multiple motivations, we focus on justifying our approach as a promising alternative in improving the performance of a full parser. First, we review the supertagging task and its applications. Because supertagging requires the existence of a particular TAG, we subsequently introduce automatically extracted TAGs and motivate their use. Although extracted grammars have their advantages, supertagging using automatically extracted TAGs runs into damaging sparse data problems. We review n-best supertagging as one means of alleviating these problems. Lastly, we run experiments that show supertagging is potentially a viable option in order to speed up a full parser. Throughout this section, we describe the kinds of linguistic resources that we use in all of our experiments and the kinds of notation that we will employ in the rest of this paper.

2.1. Supertagging

Supertagging (Bangalore and Joshi, 1999) is the process of assigning the best TAG elementary tree, or supertag, to each word in the input sentence. It performs the task of parsing disambiguation to such an extent that it may be characterized as providing an almost parse. There exist linear time approaches to supertagging, providing one promising route to linear time parsing disambiguation. However, Srinivas (1997) shows that standard n-gram modeling may be used to perform supertagging with accuracy that is adequate for partial parsing, but not for full parsing. On the other hand, n-gram modeling of supertagging has been found to be useful in other applications such as information retrieval (Chandrasekhar and Srinivas, 1997b) and text simplification (Chandrasekhar and Srinivas, 1997a).

2.2. Automatically Extracted Grammars

Recently, procedures have been developed that automatically extract TAGs from broad coverage treebanks (Xia, 1999; Chen and Vijay-Shanker, 2000; Chiang, 2000). They have the advantage that linguistically motivated TAGs can be extracted from widely available treebanks without a huge investment in manual labor. Furthermore, because of their direct extraction from a treebank, parameters can be easily and accurately estimated for building statistical TAG models for parsing (Chiang, 2000; Sarkar, 2001) or generation (Bangalore, Chen and Rambow, 2001).

In our experiments, we use an automatically extracted TAG grammar similar to the ones described by Chen and Vijay-Shanker (2000). This grammar has been extracted from Sections 02-21 of the Penn Treebank (Marcus, Santorini and Marcinkiewicz, 1993). It contains 3964 tree frames (non-lexicalized elementary trees). The parameters of extraction are set as follows. Each tree frame contains nodes that are labeled using a label set similar to the XTAG (XTAG-Group, 2001) label set. Furthermore, tree frames are extracted corresponding to a "moderate" domain of locality. Also, only those empty elements in the Penn Treebank that are usually found in TAG (subject and object trace, for example) are included in this grammar.

2.3. N-best Supertagging

The efficacy of n-gram modeling of supertagging is limited by sparse data problems of very large TAGs, such as those that are automatically extracted from broad coverage treebanks. Chen and Vijay-Shanker (2000) show that supertagging using extracted TAGs is performed at a lower accuracy (around 80%) than accuracies that have been published for supertagging using hand-written TAGs (around 90%). Faced with this evidence, it might seem that it is a hopeless task to use supertagging using extracted TAGs as a preprocessing step to accelerate full parsing. On the other hand, Chen, Bangalore and Vijay-Shanker (1999) investigate class-based supertagging, a variant of supertagging where a small set of supertags are assigned to each word instead of a single supertag. The idea is to make the sets small enough to represent a significant reduction in ambiguity so as to speed up a full parser, but to construct the sets so that class-based supertagging is much more accurate than supertagging.

One such promising class-based supertagging model is n-best supertagging, where a trigram model assigns up to n supertags for each word of the input sentence. Let $W = w_1, \ldots, w_n$ represent the sequence of words that is the input to a supertagger. Let $T_{tri} = t_{1,1}, \ldots, t_{n,1}$ be the output of the (1-best) trigram supertagger. The output of the n-best supertagger is a sequence of n-best supertags NBEST$(i) = t_{i,1}, \ldots, t_{i,n(i)}$ for each word w_i such that each supertag $t_{i,j}$ has an associated confidence score $c_{i,j}$. Assume that each sequence NBEST(i) is sorted in descending order according to these confidence scores.

The n-best supertagger is obtained by a modification of the (1-best) trigram model of supertagging. Both supertaggers first use the Viterbi algorithm to find T_{tri} by computing the most likely path $p(T_{tri})$ through a lattice of words and pairs of supertags. In the trigram supertagger, each node k along the path $p(T_{tri})$ is associated with exactly one prefix probability (the highest). In contrast, the corresponding node k in the n-best supertagger is associated with the n highest prefix probabilities. This difference allows the n-best supertagger to associate up to n supertags for each word w_i. The confidence score $c_{i,j}$ of supertag $t_{i,j}$ is the jth-best prefix probability of a node k divided by the least best prefix probability of the same node.

2.4. Parsing with N-best Supertagger Output

We claim that supertagging is a viable option to explore for use as a preprocessing step in order to speed up full parsing. In order to substantiate this claim, we perform exploratory experiments that show the relationship between n-best supertagging and parsing performance. Using the grammar that is described in Section 2.2, we train n-best supertaggers on Sections 02-21 of the Penn Treebank. For each supertagger, we supertag Section 22, which consists of about 40,100 words in 1,700 sentences. We then feed the resulting output through the LEM parser, a head-driven TAG chart parser (Sarkar, 2000). Given an input sentence and a grammar, this parser either outputs nothing, or a packed derivation forest of every parse that can be assigned to the sentence by the grammar. It does not return partial parses.

The results of these experiments are shown in Table 1. The input to the parser can be the output of either a 1, 2, or 4-best supertagger. It can also be sentences where each word is associated with all of the supertags with that word's part of speech, as determined by a trigram part of speech tagger. This is labeled as "POS-tag" in the table. Lastly, it can simply be sentences where each word is associated with the correct supertag. This is labeled as "Key." The table shows the supertagging accuracy of each corpus that is input to the parser. It also shows each

Table 1: Relationships between n-best supertagging and parsing

Input to Parser	% Supertagging Accuracy	Ambiguity (supertags/word)	% Sentences Receiving Some Parse	Time to Parse Corpus
1-best	81.47	1.0	28.2	$<$ 3 hours
2-best	88.36	1.9	53.6	$<$ 2 days
4-best	91.41	3.6	76.7	2-3 weeks
8-best	92.77	6.3	-	-
POS-tag	97.30	441.3	-	-
Key	100.00	1.0	97.0	$<$ 5 hours

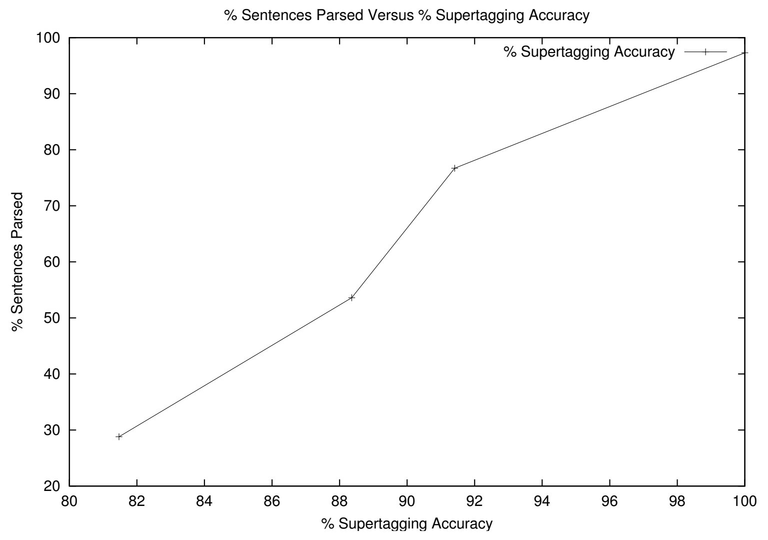

Figure 1: Percentage of Sentences That Were Parsed Versus Percent Supertagging Accuracy

corpus's ambiguity in supertags per word, the percentage of sentences in the corpus which the parser successfully found a parse, and also the time to parse the corpus. Parsing results are not available for "8-best" and "POS-tag" because of the unreasonable amount of time the parser takes for those kinds of corpora.

Table 1 reveals some interesting aspects of the relationship between supertagging and parsing. For example, it shows that merely doing part of speech tagging is inadequate as a preprocessing step if the purpose is to significantly speed up full parsing. In contrast, it also shows that the 1-best supertagger does speed up full parsing, but at the cost of missing many parses of sentences. Row "Key" shows that if supertagging works accurately enough, then it would indeed fulfill the promise of speeding up a full parser.

The second column of Table 1 is plotted against its fourth column in Figure 1. It shows how the percentage of parsed sentences in the test corpus increases as the supertagging accuracy on the test corpus increases. There is the obvious result that a higher supertagging accuracy always leads to a greater percentage of sentences being able to be parsed. There is apparently a less obvious result that this relationship is non-linear; the steepest increase in percentage of parseable sentences occurs for supertagging accuracies between 88% and 92%.

We have seen that full parsing of automatically extracted TAG grammars is apparently quite slow. We have also seen that simply part of speech tagging the input sentences as a preprocessing step does not seem to reduce ambiguity to a sufficient degree in order to speed up full parsing to a desirable extent. On the other hand, we have shown that 1-best supertagging does indeed speed up full parsing considerably—at least more than tenfold. However, in order for supertagging to fully parse a considerable portion of a corpus, it is necessary to achieve sufficiently high supertagging accuracies. Regarding the use of n-best supertagged input to a parser, we have seen that it is best to keep $n \leq 3$ in order to prevent extreme degradation in parsing performance.

2.5. Summary

We have seen that reranking the output of an n-best supertagger based on a TAG extracted from a treebank is attractive for a variety of reasons. Use of such a TAG is justified because parameters for stochastic models can be estimated easily and accurately. Use of an n-best supertagger is justified because of the considerable potential error reduction and its implications. In particular, it can be clearly seen from Table 1 that an optimal reranking of the output of an 8-best supertagger would achieve a more than 50% reduction in supertagging error. It is not unreasonable to believe that this would greatly improve the performance of applications based on supertagging, such as information retrieval and text simplification. Furthermore, Figure 1 shows that this error reduction would greatly increase the viability of using supertagging as a preprocessing step to speed up parsing.

3. Reranking an N-Best Supertagger

Our reranker takes as input a set of sentences that has been supertagged by an 8-best supertagger, including a confidence score for each selected supertag. It then ranks them according to its model. This model is trained using the machine learning program RankBoost (Freund *et al.*, 1998) which learns from sets of correctly supertagged sentences the same sentences that have been supertagged using an 8-best supertagger.

We use the variant of RankBoost introduced by (Collins, 2000). Further information about RankBoost is found in (Schapire, 1999). RankBoost learns from a set of examples. For our purpose, an example is an occurrence of a word w_i in a particular sentence along with its supertag $t_{i,j}$ selected by an n-best supertagger and its confidence score $c_{i,j}$. Each example is associated with a set of m binary indicator functions $h_s(t_{i,j})$ for $1 \leq s \leq m$. For example, UNI(w,s) is a two-argument feature template that states that the current word w has supertag s. When this template is instantiated with w_i =book and $t_{i,j} = \alpha$NXN, we obtain the following indicator function: function might be

$$h_{1234}(t_{i,j}) = \begin{cases} 1 & if\ t_{i,j} = \alpha NXN\ and\ w_i = book \\ 0 & otherwise \end{cases} \tag{1}$$

Each indicator function h_s is associated with its own parameter α_s. There is also a parameter α_0 associated with the confidence score. Training is a process of setting the parameters α to minimize the loss function:

$$loss(\alpha) = \sum_{i,j} e^{-\left(\alpha_0(ln(c_{i,1}) - ln(c_{i,j})) + \sum_s \alpha_s(h_s(t_{i,1}) - h_s(t_{i,j}))\right)} \tag{2}$$

At the start of training, no features are selected, i.e., all of the α_s's are set to zero. The optimization method that is used in training is greedy; at each iteration it picks a feature h_s which has the most impact on the loss function. The result is a set of indicator functions whose output on a given candidate is summed. These sums are used to rerank a set of candidates. Another set of examples—tuning data—is used to choose when to stop.

4. Initial Experiments

A set of features is required in order to train RankBoost to rerank supertags. As pointed out by Srinivas (1997), the traditional n-gram modeling of supertagging suffers from the flaw of only considering local dependencies when deciding how to supertag a given word. This is counter to one of the attractions of the TAG formalism, namely that even long distance dependencies are localized within a given TAG (Schabes, Abeillé and Joshi, 1988). Chen, Bangalore and Vijay-Shanker (1999) provide an example sentence where non-local context is needed to determine the correct supertag: “Many Indians *feared* their country *might* split again.” Here, the supertag for the word *feared* is partially determined by the proximity of the word *might*. Chen, Bangalore and Vijay-Shanker (1999) introduce the notion of *head supertag* context which they show increases supertagging accuracy when suitably folded into a stochastic model. While the notion of head supertags can be useful, it cannot be straightforwardly applied to our current situation; determining head supertags was feasible in (Chen, Bangalore and Vijay-Shanker, 1999) because they used the XTAG grammar, whereas it is not immediately clear which supertags should be head supertags in our extracted grammar, which is an order of magnitude larger than the XTAG grammar (3964 tree frames in the extracted grammar versus 500 tree frames in the XTAG grammar).

Chen, Bangalore and Vijay-Shanker (1999) make it clear, however, that both local and long distance features are important. In that spirit, we have designed an initial set of feature templates that is shown in Table 2. For example, UNI is a two-argument feature template that states that the current word w_0 has the supertag $t_{0,1}$. Feature

Table 2: Feature Templates Used In Initial Experiments

w_i	ith word in input sentence relative to current word which is w_0
t_i	supertag of ith word in input sentence relative to current word which is w_0

Name	Parameter List	Example of Instantiation
UNI	$w_0, t_{0,j}$	$w_0 =$book$, t_{0,j} = \alpha$NXN
BI	$w_0, t_{-1,1}, t_{0,j}$	$w_0 =$book$, t_{-1,1} = \beta$Nn$, t_{0,j} = \alpha$NXN
TRI	$w_0, t_{-2,1}, t_{-1,1}, t_{0,j}$	$w_0 =$book$, t_{-2,1} = \beta$Dnx$, t_{-1,1} = \beta$Nn$, t_{0,j} = \alpha$NXN
FORWARD-BI	$w_0, t_{0,j}, t_{1,1}$	$w_0 =$book$, t_{0,j} = \alpha$NXN$, t_{1,1} = \alpha$nx0V
FORWARD-TRI	$w_0, t_{0,j}, t_{1,1}, t_{2,1}$	$w_0 =$book$, t_{0,j} = \alpha$NXN$, t_{1,1} = \alpha$nx0V$, t_{2,1} = \beta$vxN
LEFT-FAR-BI$_x$ ($3 \leq x \leq 8$)	$t_{-x,1}, t_{0,j}$	$t_{-x,1} = \beta$Dnx$, t_{0,j} = \alpha$NXN
RIGHT-FAR-BI$_x$ ($3 \leq x \leq 8$)	$t_{0,j}, t_{x,1}$	$t_{0,j} = \alpha$NXN$, t_{x,1} = \beta$nxPnx
LEFT-WIN$_x$ ($x \in$ { 4, 8, 16 })	$t_{-y,1}, t_{0,j}$	$t_{-y} = \alpha$nx0Vnx1, $0 < y \leq x$, $t_{0,j} = \alpha$NXN
RIGHT-WIN$_x$ ($x \in$ { 4, 8, 16 })	$t_{0,j}, t_{y,1}$	$t_{0,j} = \alpha$NXN$, t_{y,1} = \beta$nxPnx, $0 < y \leq x$

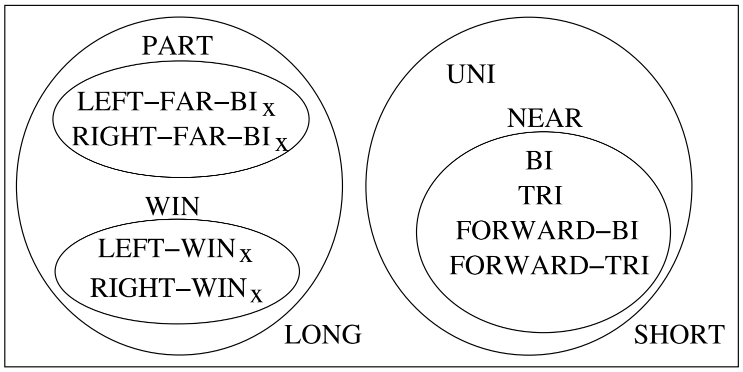

Figure 2: Sets of Features That Are Used In Various Experiments

templates exist that take into account local context and others that take into account long distance context. Local feature templates basically take into consideration the same context that a trigram model considers. They are UNI, BI, TRI, FORWARD-BI, and FORWARD-TRI. Long distance feature templates take into consideration extra-trigram context. There are two kinds of long distance feature templates: *-FAR-BI$_x$ and *-WIN$_x$. The *-FAR-BI$_x$ kind states that the current word has the supertag $t_{0,j}$ and there exists a supertag a *fixed* distance x away from the current word having supertag $t_{x,1}$. The *-WIN$_x$ kind of feature template states that the current word has the supertag $t_{0,j}$ and there exists a supertag $t_{y,1}$ which lies *within some distance y*, $0 < y \leq x$, of the current word.

The list of feature templates in Table 2 is somewhat long and unwieldy. In order to simplify our exposition of different reranking models, we have given names to various subsets of these feature templates. These are shown in Figure 2. The set of all *-FAR-BI$_x$ feature templates is called PART. The set of all *-WIN$_x$ feature templates is called WIN. PART$\cup$WIN yields LONG. SHORT is the set of all trigram-context feature templates. NEAR is SHORT – UNI.

Training RankBoost for reranking supertags requires n-best supertagged data. This is obtained by first extracting a TAG from the Penn Treebank as described in Section 2.2. 8-best supertaggers are then used to derive training, tuning, and test data. Ten-fold cross validation of Sections 02-11 and part of 12 provides the training data (475197 words). 8-best supertagged versions of the rest of Section 12 and Sections 13-14 serve as tuning data (94975 words). Test data is derived from the output of an 8-best supertagger trained on Sections 02-14 on Section 22 (40117 words). Note that for these experiments, a truncated version of the usual Penn Treebank training data–Sections 02-21, are used. This is done merely to expedite the training and testing of different reranking models.

Table 3 shows the supertagging accuracy results for the n-best supertagger, before and after reranking by

Table 3: N-best supertagger results and Reranker results using different feature sets on Section 22.

	% Supertag Accuracy						
n-best	Before Rerank	SHORT	LONG	LONG ∪ SHORT	LONG ∪ UNI	WIN ∪ UNI	PART ∪ UNI
1	80.20	80.77	80.13	81.73	81.39	81.63	81.04
2	87.13	87.67	87.13	88.59	88.38	88.55	88.09
3	89.24	89.73	89.24	90.24	90.16	90.25	89.88
4	90.28	90.63	90.28	90.95	90.88	90.98	90.77
5	90.84	91.07	90.83	91.33	91.27	91.33	91.19
6	91.22	91.38	91.20	91.54	91.50	91.54	91.44
7	91.52	91.57	91.52	91.66	91.64	91.65	91.62
8	91.73	91.73	91.73	91.73	91.73	91.73	91.73

RankBoost. The n-best results for $1 \leq n < 8$ are derived by considering only the top n supertags proposed by the 8-best supertagger. The left half of the table shows three different models are trained using RankBoost, one that uses SHORT features only, one that uses LONG features only, and another that uses both LONG and SHORT features. The rules that are learned by RankBoost are then applied to the 8-best supertags to rerank them.

The results are encouraging. The 1-best supertagger achieves an accuracy of only 80.20%. Nevertheless, the 8-best accuracy is 91.73% which shows that an optimal reranking procedure would halve the error rate. Reranking using SHORT features results in a statistically significant error reduction (p <0.05) of 2.9% for 1-best. Reranking also using LONG features results in an error reduction of 7.7% for 1-best (and an error reduction of 13.3% with respect to the RankBoost topline of 91.73%). Therefore RankBoost is obviously able to use LONG features effectively in conjunction with the SHORT features, despite a big increase in the number of parameters of the model. Note also that reranking improves the accuracy for all n-best results, $1 \leq n < 8$.

Apparently, there is some interaction between LONG and SHORT features which makes model LONG∪SHORT effective whereas model LONG is useless. In order to study this interaction, and also to determine what kinds of LONG features help the most, we have tested models LONG∪UNI, WIN∪UNI, and PART∪UNI. The results are shown in the right half of Table 3. Model LONG∪UNI achieves much of the performance of model LONG∪SHORT, even though it only considers the unigram feature. One possible explanation for this phenomenon is that SHORT features aid LONG features not because the local trigram context that is modeled by SHORT is so much more important, but instead it is lexicalization that is important, SHORT features being lexicalized whereas LONG features are not. Also note that model WIN∪UNI outperforms model PART∪UNI. This seems to indicate that PART feature templates are less useful in supertag disambiguation than WIN feature templates.

5. Analysis of Some Initial Experiments

At first glance, there does not seem to be much of a difference between model LONG∪SHORT and model SHORT. The difference between them in terms of accuracy of 1-best supertagging reranking is slightly less than one percent, about five percent in terms of reduction in error. On the other hand, as Table 6 shows, this small difference is still statistically significant. In order to get a better grasp on the differences in behavior of model LONG∪SHORT and model SHORT, and also to get a feeling about how one might improve reranking models for supertagging, we perform a semi-qualitative analysis of the 1-best reranked output of these two models.

The ten most frequently mistagged supertags (i.e. those supertags that were most misclassified by the reranker), sorted by frequency, for model SHORT and model LONG∪SHORT are shown in Table 4. At first glance, there is not much difference between the two models; they both mistag mostly the same kinds of supertags, and the supertags' rankings are about the same. However, certain differences can be discerned. Notably, the frequency of mistagging αNXN is 25% less in LONG∪SHORT than it is in SHORT. Also, there is somewhat less of a PP attachment problem in LONG∪SHORT than there is in SHORT, as can be seen by the frequencies of the PP attachment supertags βnxPnx and βvxPnx. The fact that the frequency of mistaggings of αnx0Vnx1 drops from 168 in SHORT to 130 in LONG∪SHORT is also noteworthy; apparently LONG∪SHORT is performing better at resolving NP versus S subcat ambiguity.

For each of several supertags in Table 4, we proceed to determine the most important features that

Table 4: Ten Most Frequently Mistagged Supertags, By Frequency, for SHORT and LONG∪SHORT

SHORT				LONG∪SHORT			
Frequency	Supertag	Frequency	Supertag	Frequency	Supertag	Frequency	Supertag
650	αNXN	167	βnxN	474	αNXN	155	βVnx
410	βNnx	162	βnxPunct	356	βNnx	151	βnxPnx
303	βvxPnx	148	βnxPnx	289	βvxPnx	147	αN
216	βAnx	130	βucpPunct	203	βAnx	144	βnxPunct
168	αnx0Vnx1	117	αN	166	βnxN	130	αnx0Vnx1

LONG∪SHORT uses in order to choose the correct supertag. Our methodology is as follows. Given a supertag γ, we determine the set of instances in the test corpus where LONG∪SHORT reranked γ to first place from an originally lower ranking. For each instance, we determine the features that caused LONG∪SHORT to rank γ more highly, tabulating the number of times each feature is used. We also record the multiset $\phi(\gamma)$ of supertags $\gamma' \neq \gamma$ such that LONG∪SHORT replaced γ' with γ as the first ranked supertag.

Consider supertag αnx0Vnx1. Most frequently occurring members of ϕ(αnx0Vnx1) include βVvx, βnx0Vs1, αINnx0Vnx1 (declarative transitive supertag with complementizer), and βvxINnx0Vnx1. The most frequently used features that are used to rank αnx0Vnx1 more highly are LEFT-WIN$_{16}$(EOS,αnx0Vnx1) and LEFT-WIN$_{8}$(EOS,αnx0Vnx1), where EOS is a sentence delimiter, in this case the left sentence delimiter. Intuitively, these features seem to suggest that αnx0Vnx1 should appear nearer to the beginning of the sentence than for example, βvxINnx0Vnx1, being a verbal postmodifier, should. Another frequently used feature is LEFT-WIN$_{4}$(αNXN,αnx0Vnx1). It is apparently used to make sure there exists an NP to the left of the current word that would fit in the subject slot of αnx0Vnx1. The existence of the frequently used feature LEFT-WIN$_{16}$(βMDvx,αnx0Vnx1) is also of interest. Apparently, this feature occurs because αnx0Vnx1 often serves as the sentential complement of another verb to its left. This verb can take a variety of supertags, including βnx0Vs1 and βN0nx0Vs1 for example. Having a separate feature for each of these supertags would possibly lead to suboptimal reranking performance because of sparse data. Instead, apparently based on the generalization that these supertags are usually modified by a modal verb βMDvx, RankBoost chooses the feature LEFT-WIN$_{16}$(βMDvx,αnx0Vnx1).

All of the features that we have discussed are LONG. In fact, there is a preponderance of LONG features used to rank αnx0Vnx1: the ten most frequent features are LONG. There are however, some SHORT features that are heavily weighted, although they are not used quite as often. One notable SHORT feature is FORWARD-BI(has,βVvx,βDnx). Intuitively, it resolves the ambiguity between βVvx and αnx0Vnx1 by seeing whether an NP (prefixed by a determiner) immediately follows the current word.

Supertag αNXN presents another interesting case. The most frequently occurring members of ϕ(αNXN) include αnx0N, βnxN, and βvxN. The most frequently used features that are used to prefer αNXN include LEFT-WIN$_{16}$(αNXN,αNXN), RIGHT-WIN$_{16}$(αNXN,βsPeriod), RIGHT-WIN$_{16}$(αNXN,βnxPnx), LEFT-WIN$_{4}$(βNn,αNXN). These features seem to encode the context that is likely to surround αNXN. Of course, these features also seem likely to surround other members of ϕ(αNXN). Perhaps these features are chosen because of a general bias that the n-best supertagger has against supertagging head nouns appropriately.

6. Further, Exploratory Experiments

Based on our experience with reranking of n-best supertags, we have drawn some possible avenues for improvement of the reranking procedure. In the following, we list some common reasons for lack of optimum reranking performance and discuss how they might be eliminated.

- The feature that would perform the appropriate reranking is not chosen because of sparse data. Note that the supertags that do instantiate feature templates tend to be very common. It is not surprising, therefore, that there exists a feature such as LEFT-WIN$_{4}$(αNXN,αnx0Vnx1). Recall that this appears to ensure that αnx0Vnx1 has an NP subject to the left. An analogous feature is not likely to appear for an infrequently occurring supertag, such as βN0nx0Vs1. One possible solution would be to instantiate feature templates with certain aspects of supertags instead of entire supertags. Along this line, we perform some exploratory experiments in Section 6.1.

- The correct supertag for word w_0 does not exist in the n-best supertagged output. One way to ameliorate this problem is to improve the performance of the first stage n-best trigram supertagger. Along this line, we perform some exploratory experiments in Section 6.2.

- For words other than the current word, the feature template is instantiated only from the 1-best supertag output, which is not always correct. For example, the feature LEFT-WIN$_4$(αNXN,αnx0Vnx1), depends on the fact that the supertags to the immediate left of the current word are, in fact, correctly supertagged, whereas they are only correctly supertagged about 80% of the time. Now, the training process should compensate for this somewhat because the inputs to the training process are (flawed) supertagged sentences. On the other hand, perhaps a different approach would be more effective in tackling this problem. One avenue would be to rerank n-best *paths* of supertags, instead of n-best per word supertagged output. Along this line, we have implemented an n-best paths supertagger, based on a trigram model, but employing a search strategy similar to (Ratnaparkhi, 1996). Trained on Sections 02-21 of the Penn Treebank, this supertagger achieves about 89% supertag accuracy only when the top 100 paths are chosen. It remains to be seen whether this will cause difficulties in terms of memory or space resources for training the reranker.

6.1. Training the Reranker with Part of Speech Features

Having features consider entire supertags is a limiting factor in contributing to the performance of the reranker not in the least because of sparse data. One possible solution is to base features on aspects of supertags instead of entire supertags. For example, one might take the approach of breaking down each supertag into a feature vector (Srinivas, 1997; Xia *et al.*, 1998; Xia *et al.*, 2000; Chen, 2001), and to base RankBoost features on elements of that vector. Another approach would be to consider each supertag as generated by a Markov process (Collins, 1997; Collins, 1999). In this case, one would base RankBoost features on individual steps in that process. Here, we consider using part of speech as a component in feature space.

As implied by Section 2.2, the preterminal tag set for our extracted grammar is similar to the XTAG part of speech tag set. For our features, we can either choose to retain the XTAG parts of speech or use the more detailed Penn Treebank part of speech tagset. This choice displays the usual tradeoff between assuaging sparse data (the former) and having detailed enough features to make appropriate decisions (the latter). We have chosen the latter because the Penn Treebank part of speech tagset (about 45 tags) is already an order of magnitude smaller than the supertag tagset (about 3900 tags), although we believe that it would also be interesting to repeat our experiments using the XTAG part of speech tagset (about 20 tags).

For each feature template in LONG∪SHORT, an analogous feature template is created with supertag parameters other than the current word replaced with part of speech parameters. For example, LEFT-WIN$_4$-POS(p_y,$t_{0,cur}$) is a feature template that states that the current word is supertagged with $t_{0,cur}$ and there exists a word to the left that has part of speech p_y within a distance of four words of the current word. Furthermore, we give the same name to these new subsets of feature templates as is given to the previous subsets, affixed with -POS. For example, WIN-POS is the set of feature templates consisting of LEFT-WIN$_x$-POS and RIGHT-WIN$_x$-POS.

After the RankBoost training, tuning, and test corpora were suitably annotated using a trigram model Penn Treebank part of speech tagger, models NEAR-POS∪SHORT and LONG-POS∪LONG∪SHORT were trained and tested. The results are shown in the left half of Table 5. Although the 1-best reranking accuracies are not significantly higher for the *-POS models than for the corresponding non-POS models (Table 6), it is important to keep in mind that these are preliminary results. We believe that the higher accuracies for the *-POS models indicate that there may exist other, untried models which use part of speech information more effectively.

6.2. Reranking of Smoothed N-Best Supertagging

There are many cases where the reranker cannot give the correct supertag the top ranking because it does not exist in the n-best output. One possible solution to this problem is to enhance the n-best supertagger by smoothing its emit probability $p(w|t)$, and then run the reranker on the resulting output. Here, we perform such an experiment.

Our experiment proceeds as follows. We choose to smooth $p(w|t)$ using the approach mentioned in (Chen, 2001). It accounts especially for the fairly large set of cases (about 5%) in which the word and the correct supertag have both been seen in the training data, but not in combination. These cases would normally be assigned a probability of zero by the supertagging model. Using this approach, we prepared training, tuning, and test data using the smoothed version of the n-best supertagger as appropriate. We subsequently trained model LONG∪SHORT on this training and tuning data, and then tested the reranker as usual.

Table 5: N-best supertagger results, smoothing, and smoothing plus LONG ∪ SHORT reranker results

	% Supertag Accuracy				
n-best	Before Rerank	NEAR-POS ∪ SHORT	LONG-POS ∪ LONG∪SHORT	Smoothed	Smoothed and LONG∪SHORT
1	80.20	80.97	82.04	81.64	82.99
2	87.13	87.77	88.83	89.02	90.42
3	89.24	89.77	90.38	91.24	92.31
4	90.28	90.63	91.04	92.37	93.14
5	90.84	91.07	91.34	93.07	93.59
6	91.22	91.37	91.53	93.54	93.88
7	91.52	91.57	91.65	93.84	94.05
8	91.73	91.73	91.73	94.14	94.14

Table 6: Differences in 1-best supertagging accuracy for all pairs of reranking models. Significant differences ($p < 0.05$) are marked with "*"

	Before Rerank	S	L	L ∪ S	L ∪ U	W ∪ U	P ∪ U	SM	SM & L ∪ S	NPOS ∪ S
S	+0.57									
L	-0.07	-0.64								
L ∪ S	+1.53*	+0.96*	+1.60*							
L ∪ U	+1.19*	+0.62	+1.26*	-0.34						
W ∪ U	+1.43*	+0.86	+1.50*	-0.10*	+0.24					
P ∪ U	+0.84	+0.27	+0.91*	-0.69	-0.35	-0.59				
SM	+1.44*	+0.87	+1.51*	-0.09	+0.25	+0.01	+0.60			
SM & L ∪ S	+2.79*	+2.22*	+2.86*	+1.26*	+1.60*	+1.36*	+1.95*	+1.35*		
NPOS ∪ S	+0.77	+0.20	+0.84	-0.76	-0.42	-0.66	-0.07	-0.67	-2.02*	
LPOS ∪ L ∪ S	+1.84*	+1.27*	+1.91*	+0.31	+0.65	+0.41	+1.00*	+0.40	-0.95*	+1.07*

The smoothing technique was successful in raising the 8-best supertagging accuracy to 94.14% from 91.73%. And, as can be seen in Table 5 RankBoost can still improve on the output, though to a slightly lesser extent. Overall, the error reduction increases to 14.1% over the unsmoothed, non-reranked 1-best supertags (of which RankBoost contributes 6.9% absolute). As far as we know, these are the currently best results for supertagging using large supertag sets.

7. Significance Testing

We performed a one-way analysis of variance on the 1-best supertagging results of all of the reranking models that are mentioned in this paper. Table 6 tabulates the differences between 1-best supertagging accuracies of the various models and marks significant differences, $p < 0.05$, with "*." The F-value is 18.11; the critical value for the Tukey test is 0.89.

8. Conclusions and Future Work

This paper has explored the use of RankBoost in order to rerank an n-gram supertagger. We have seen that such a reranking, performed effectively, is potentially useful in a variety of applications, including speeding up a parser. We have performed experiments that show that RankBoost can indeed produce models that perform reranking well, to a statistically significant degree. We have identified specific features that explain why the reranker performs effectively. We have also identified causes that limit the reranker's performance. Finally, we have performed other, exploratory experiments that ameliorate these limitations.

An advantage of using RankBoost is that numerous candidate features can be added robustly because RankBoost learns to choose only the relevant ones. This invites the possibility of investigating kinds of features for reranking other than the ones mentioned in this paper. Bilexical features may be useful, along with features that take into account tree families, different kinds of parts of speech, punctuation, or the results of chunkers or even parsers. It is also important to keep in mind that the performance of the reranker is limited by the performance of the n-best supertagger. Thus, novel means to increase the n-best supertagger's accuracy should also be explored. We also intend to investigate other ways of obtaining candidate supertag sets using other notions of class-based supertagging presented in (Chen, Bangalore and Vijay-Shanker, 1999).

References

Bangalore, Srinivas, John Chen and Owen Rambow. 2001. Impact of Quality and Quantity of Corpora on Stochastic Generation. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Langauge Processing*, Pittsburgh, PA.

Bangalore, Srinivas and A. K. Joshi. 1999. Supertagging: An Approach to Almost Parsing. *Computational Linguistics*, 25(2).

Chandrasekhar, R. and B. Srinivas. 1997a. Automatic Induction of Rules for Text Simplification. *Knowledge-Based Systems*, 10:183–190.

Chandrasekhar, R. and B. Srinivas. 1997b. Using Supertags in Document Filtering: The Effect of Increased Context on Information Retrieval. In *Proceedings of Recent Advances in NLP '97*.

Chen, John. 2001. *Towards Efficient Statistical Parsing Using Lexicalized Grammatical Information*. Ph.D. thesis, University of Delaware.

Chen, John, Srinivas Bangalore and K. Vijay-Shanker. 1999. New Models for Improving Supertag Disambiguation. In *Proceedings of the 9th Conference of the European Chapter of the Association for Computational Linguistics*, Bergen, Norway.

Chen, John and K. Vijay-Shanker. 2000. Automated Extraction of TAGs from the Penn Treebank. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pages 65–76.

Chiang, David. 2000. Statistical Parsing with an Automatically-Extracted Tree Adjoining Grammar. In *Proceedings of the the 38th Annual Meeting of the Association for Computational Linguistics*, pages 456–463, Hong Kong.

Collins, Michael. 1997. Three Generative Lexicalized Models for Statistical Parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*.

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

Collins, Michael. 2000. Discriminative Reranking for Natural Language Parsing. In *Proceedings of the 17th International Conference on Machine Learning*.

Freund, Yoav, Raj Iyer, Robert E. Schapire and Yoram Singer. 1998. An Efficient Boosting Algorithm for Combining Preferences. In *Machine Learning: Proceedings of the Fifteenth International Conferece*.

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

Ratnaparkhi, Adwait. 1996. A Maximum Entropy Model for Part-of-Speech Tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, Somerset, NJ.

Sarkar, Anoop. 2000. Practical Experiments in Parsing using Tree Adjoining Grammars. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammars and Related Frameworks*, Paris, France.

Sarkar, Anoop. 2001. Applying Co-Training Methods to Statistical Parsing. In *Proceedings of Second Annual Meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, PA.

Schabes, Yves, Anne Abeillé and Aravind K. Joshi. 1988. Parsing Strategies with 'Lexicalized' Grammars: Application to Tree Adjoining Grammars. In *Proceedings of the 12th International Conference on Computational Linguistics*, Budapest, Hungary.

Schapire, Robert E. 1999. A Brief Introduction to Boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*.

Srinivas, B. 1997. Performance Evaluation of Supertagging for Partial Parsing. In *Proceedings of the Fifth International Workshop on Parsing Technologies*, pages 187–198, Cambridge, MA.

van Halteren, H. 1996. Comparison of Tagging Strategies: A Prelude to Democratic Tagging. In *Research in Humanities Computing 4*. Clarendon Press, Oxford, England.

Walker, Marilyn A., Owen Rambow and Monica Rogati. 2001. SPoT: A Trainable Sentence Planner. In *Proceedings of the Second Meeting of the North American Chapter of the Association for Computational Linguistics*, pages 17–24.

Xia, Fei. 1999. Extracting Tree Adjoining Grammars from Bracketed Corpora. In *Fifth Natural Language Processing Pacific Rim Symposium (NLPRS-99)*, Beijing, China.

Xia, Fei, Chung hye Han, Martha Palmer and Aravind Joshi. 2000. Comparing Lexicalized Treebank Grammars Extracted from Chinese, Korean, and English Corpora. In *Proceedings of the Second Chinese Language Processing Workshop (CLP-2000)*, Hong Kong, China.

Xia, Fei, Martha Palmer, K. Vijay-Shanker and Joseph Rosenzweig. 1998. Consistent Grammar Development Using Partial-Tree Descriptions for Lexicalized Tree Adjoining Grammars. In *Fourth International Workshop on Tree Adjoining Grammars and Related Frameworks*, pages 180–183.

XTAG-Group, The. 2001. A Lexicalized Tree Adjoining Grammar for English. Technical report, University of Pennsylvania. Updated version available at http://www.cis.upenn.edu/˜xtag.