Boosting Applied to Tagging and PP Attachment

Steven Abney **Robert E. Schapire** **Yoram Singer**

AT&T Labs − Research
180 Park Avenue
Florham Park, NJ 07932
{abney, schapire, singer}@research.att.com

Abstract

Boosting is a machine learning algorithm that is not well known in computational linguistics. We apply it to part-of-speech tagging and prepositional phrase attachment. Performance is very encouraging. We also show how to improve data quality by using boosting to identify annotation errors.

1 Introduction

Boosting is a machine learning algorithm that has been applied successfully to a variety of problems, but is almost unknown in computational linguistics. We describe experiments in which we apply boosting to part-of-speech tagging and prepositional phrase attachment. Results on both PP-attachment and tagging are within sampling error of the best previous results.

The current best technique for PP-attachment (backed-off density estimation) does not perform well for tagging, and the current best technique for tagging (maxent) is below state-of-the-art on PP-attachment. Boosting achieves state-of-the-art performance on both tasks simultaneously.

The idea of boosting is to combine many simple "rules of thumb," such as "the current word is a noun if the previous word is *the*." Such rules often give incorrect classifications. The main idea of boosting is to combine many such rules in a principled manner to produce a single highly accurate classification rule.

There are similarities between boosting and transformation-based learning (Brill, 1993): both build classifiers by combining simple rules, and both are noted for their resistance to overfitting. But boosting, unlike transformation-based learning, rests on firm theoretical foundations; and it outperforms transformation-based learning in our experiments.

There are also superficial similarities between boosting and maxent. In both, the parameters are weights in a log-linear function. But in maxent, the log-linear function defines a probability, and the objective is to maximize likelihood, which may not minimize classification error. In boosting, the log-linear function defines a hyperplane dividing examples into (binary) classes, and boosting minimizes classification error directly. Hence boosting is usually more appropriate when the objective is classification rather than density estimation.

A notable property of boosting is that it maintains an explicit measure of how difficult it finds particular training examples to be. The most difficult examples are very often mislabelled examples. Hence, boosting can contribute to improving data quality by identifying annotation errors.

2 The boosting algorithm AdaBoost

In this section, we describe the boosting algorithm AdaBoost that we used in our experiments. AdaBoost was first introduced by Freund and Schapire (1997); the version described here is a (slightly simplified) version of the one given by Schapire and Singer (1998). A formal description of AdaBoost is shown in Figure 1. AdaBoost takes as input a training set of m labeled examples $\langle (x_1, y_1), \ldots, (x_m, y_m) \rangle$ where x_i is an example (say, as described by a vector of attribute values), and $y_i \in \{-1, +1\}$ is the label associated with x_i. For now, we focus on the binary case, in which only two labels (positive or negative) are possible. Multiclass problems are discussed later.

Formally, the rules of thumb mentioned in the introduction are called **weak hypotheses**. Boosting assumes access to an algorithm or subroutine for generating weak hypotheses called the **weak learner**. Boosting can be combined with any suitable weak learner; the one that we used will be described below.

AdaBoost calls the weak learner repeatedly in a series of rounds. On round t, AdaBoost provides the weak learner with a set of **importance weights** over the training set. In response, the weak learner com-

Given: $(x_1, y_1), \ldots, (x_m, y_m)$
where $x_i \in X$, $y_i \in \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : X \to \mathbb{R}$.
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-y_i h_t(x_i))}{Z_t}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} h_t(x)\right).$$

Figure 1: The boosting algorithm AdaBoost.

putes a weak hypothesis h_t that maps each example x to a real number $h_t(x)$. The sign of this number is interpreted as the predicted class (-1 or $+1$) of example x, while the magnitude $|h_t(x)|$ is interpreted as the level of **confidence** in the prediction, with larger values corresponding to more confident predictions.

The importance weights are maintained formally as a distribution over the training set. We write $D_t(i)$ to denote the weight of the ith training example (x_i, y_i) on the tth round of boosting. Initially, the distribution is uniform. Having obtained a hypothesis h_t from the weak learner, AdaBoost updates the weights by multiplying the weight of each example i by[1] $e^{-y_i h_t(x_i)}$. If h_t incorrectly classified example i so that $h_t(x_i)$ and y_i disagree in sign, then this has the effect of increasing the weight on this example, and conversely the weights of correctly classified examples are decreased. Moreover, the greater the confidence of the prediction (that is, the greater the magnitude of $h_t(x_i)$), the more drastic will be the effect of the update. The weights are then renormalized, resulting in the update rule shown in the figure.

In our experiments, we used cross validation to choose the number of rounds T. After T rounds, AdaBoost outputs a **final hypothesis** which makes predictions using a simple vote of the weak hypotheses' predictions, taking into account the varying confidences of the different predictions. A new example x is classified using

$$f(x) = \sum_{t=1}^{T} h_t(x),$$

where the label predicted for x is $\text{sign}(f(x))$.

2.1 Finding weak hypotheses

In this section, we describe the weak learner used in our experiments. Since we now focus on what happens on a single round of boosting, we will drop t subscripts where possible.

Schapire and Singer (1998) prove that the training error of the final hypothesis is at most $\prod_{t=1}^{T} Z_t$. This suggests that the training error can be greedily driven down by designing a weak learner which, on round t of boosting, attempts to find a weak hypothesis h that minimizes

$$Z = \sum_{i=1}^{m} D(i) \exp(-y_i h(x_i)).$$

This is the principle behind the weak learner used in our experiments.

In all our experiments, we use very simple weak hypotheses that test the value of a Boolean predicate and make a prediction based on that value. The predicates used are of the form "$a = v$", for a an attribute and v a value; for example, "PreviousWord = *the*". In the PP-attachment experiments, we also considered conjunctions of such predicates. If, on a given example x, the predicate holds, the weak hypothesis outputs prediction p_1, otherwise p_0, where p_1 and p_0 are determined by the training data in a way we describe shortly. In this setting, weak hypotheses can be identified with predicates, which in turn can be thought of as features of the examples; thus, in this setting, boosting can be viewed as a feature-selection method.

Let $\phi(x) \in \{0, 1\}$ denote the value of the predicate ϕ on the example x, and for $b \in \{0, 1\}$, let $p_b \in \mathbb{R}$ be the prediction of the weak hypothesis when $\phi(x) = b$. Then we can write simply $h(x) = p_{\phi(x)}$. Given a predicate ϕ, we choose p_0 and p_1 to minimize Z. Schapire and Singer (1998) show that Z is minimized when we let

$$p_b = \tfrac{1}{2} \ln\left(\frac{W_{+1}^b}{W_{-1}^b}\right) \quad (1)$$

[1] Schapire and Singer (1998) multiply instead by $\exp(-y_i \alpha_t h_t(x_i))$ where $\alpha_t \in \mathbb{R}$ is a parameter that needs to be set. In the description presented here, we fold α_t into h_t.

MF tag	O	7.66	
Markov 1-gram	B	6.74	
Markov 3-gram	W	3.7	
Markov 3-gram	B	3.64	
Decision tree	M	3.5	
Transformation	B	3.39	
Maxent	R	3.37	
Maxent	O	3.11	$\pm.07$
Multi-tagger Voting	B	2.84	$\pm.03$

Table 1: TB-WSJ testing error previously reported in the literature. B = (Brill and Wu, 1998); M = (Magerman, 1995); O = our data; R = (Ratnaparkhi, 1996); W = (Weischedel and others, 1993).

for $b \in \{0,1\}$ where W_s^b is the sum of $D(i)$ for examples i such that $y_i = s$ and $\phi(x_i) = b$. This choice of p_b implies that

$$Z = 2 \sum_{b \in \{0,1\}} \sqrt{W_{+1}^b W_{-1}^b}. \quad (2)$$

This expression can now be minimized over all choices of ϕ.

Thus, our weak learner works by searching for the predicate ϕ that minimizes Z of Eq. (2), and the resulting weak hypothesis $h(x)$ predicts $p_{\phi(x)}$ of Eq. (1) on example x.

In practice, very large values of p_0 and p_1 can cause numerical problems and may also lead to overfitting. Therefore, we usually "smooth" these values using the following alternate choice of p_b given by Schapire and Singer (1998):

$$p_b = \tfrac{1}{2} \ln \left(\frac{W_{+1}^b + \varepsilon}{W_{-1}^b + \varepsilon} \right) \quad (3)$$

where ε is a small positive number.

2.2 Multiclass problems

So far, we have only discussed binary classification problems. In the multiclass case (in which more than two labels are possible), there are many possible extensions of AdaBoost (Freund and Schapire, 1997; Schapire, 1997; Schapire and Singer, 1998). Our default approach to multiclass problems is to use Schapire and Singer's (1998) AdaBoost.MH algorithm. The main idea of this algorithm is to regard each example with its multiclass label as several binary-labeled examples.

More precisely, suppose that the possible classes are $1, \ldots, k$. We map each original example x with label y to k binary labeled derived examples $(x,1), \ldots, (x,k)$ where example (x,c) is labeled $+1$ if $c = y$ and -1 otherwise. We then essentially apply binary AdaBoost to this derived problem. We maintain a distribution over pairs (x,c), treating each such as a separate example. Weak hypotheses are identified with predicates over (x,c) pairs, though they now ignore c, so that we can continue to use the same space of predicates as before. The prediction weights p_0^c, p_1^c, however, are chosen separately for each class c; we have $h_t(x,c) = p_{\phi(x)}^c$. Given a new example x, the final hypothesis makes confidence-weighted predictions $f(x,c) = \sum_{t=1}^T h_t(x,c)$ for each of the discrimination questions ($c = 1$? $c = 2$? etc.); the class is predicted to be the value of c that maximizes $f(x,c)$. For more detail, see the original paper (Schapire and Singer, 1998).

When memory limitations prevent the use of AdaBoost.MH, an alternative we have pursued is to use binary AdaBoost to train separate discriminators (binary classifiers) for each class, and combine their output by choosing the class c that maximizes $f_c(x)$, where $f_c(x)$ is the final confidence-weighted prediction of the discriminator for class c. Let us call this algorithm AdaBoost.MI (multiclass, independent discriminators). It differs from AdaBoost.MH in that predicates are selected independently for each class; we do not require that the weak hypothesis at round t be the same for all classes. The number of rounds may also differ from discriminator to discriminator.

3 Tagging

3.1 Corpus

To facilitate comparison with previous results, we used the UPenn Treebank corpus (Marcus et al., 1993). The corpus uses 80 labels, which comprise 45 parts of speech properly so-called, and 35 **indeterminate** tags, representing annotator uncertainty. We introduce an 81st label, ##, for paragraph separators.

An example of an indeterminate tag is `NN|JJ`, which indicates that the annotator could not decide between `NN` and `JJ`. The "right" thing to do with indeterminate tags would either be to eliminate them or to count the tagger's output as correct if it agrees with any of the alternatives. Previous work appears to treat them as separate tags, however, and we have followed that precedent.

We partitioned the corpus into three samples: a test sample consisting of 1000 randomly selected

	n		errors	percent	contrib	
ambig	28,557	(52.7%)	1396	4.89	2.58	
unambig	24,533	(45.3%)	167	0.68	0.31	
unknown	1104	(2.0%)	213	19.29	0.39	
total	54,194		1776		3.28	± 0.08

Table 2: Performance of the multi-discriminator approach.

paragraphs (54,194 tokens), a development sample, also of 1000 paragraphs (52,087 tokens), and a training sample (1,207,870 tokens).

Some previously reported results on the Treebank corpus are summarized in Table 1. These results are all based on the Treebank corpus, but it appears that they do not all use the same training-test split, nor the same preprocessing, hence there may be differences in details of examples and labels. The "MF tag" method simply uses the most-frequent tag from training as the predicted label. The voting scheme combines the outputs of four other taggers.

3.2 Applying Boosting to Tagging

The straightforward way of applying boosting to tagging is to use AdaBoost.MH. Each word token represents an example, and the classes are the 81 part-of-speech tags. Weak hypotheses are identified with "attribute=value" predicates. We use a rather spare attribute set, encoding less context than is usual. The attributes we use are:

- **Lexical attributes**: The current word as a downcased string (S); its capitalization (C); and its most-frequent tag in training (T). T is `unknown` for unknown words.
- **Contextual attributes**: the string (LS), capitalization (LC), and most-frequent tag (LT) of the preceding word; and similarly for the following word (RS, RC, RT).
- **Morphological attributes**: the inflectional suffix (I) of the current word, as provided by an automatic stemmer; also the last two ($S2$) and last three ($S3$) letters of the current word.

We note in passing that the single attribute T is a good predictor of the correct label; using T as the predicted label gives a 7.7% error rate (see Table 1).

Experiment 1. Because of memory limitations, we could not apply AdaBoost.MH to the entire training sample. We examined several approximations. The simplest approximation (experiment 1) is to run AdaBoost.MH on 400K training examples, instead of the full training set. Doing so yields a test error of 3.68%, which is actually as good as using Markov 3-grams (Table 1).

Exp. 1	400K training	$3.68 \pm .08$
Exp. 2	4×300K	$3.32 \pm .08$
Exp. 3	Unambiguous & definite	$3.59 \pm .08$
Exp. 4	AdaBoost.MI	$3.28 \pm .08$

Table 3: Performance on experiments 1–4.

Experiment 2. In experiment 2, we divided the training data into four quarters, trained a classifier using AdaBoost.MH on each quarter, and combined the four classifiers using (loosely speaking) a final round of boosting. This improved test error significantly, to 3.32%. In fact, this tagger performs as well as any single tagger in Table 1 except the Maxent tagger.

Experiment 3. In experiment 3, we reduced the training sample by eliminating unambiguous words (multiple tags attested in training) and indefinite tags. We examined all indefinite-tagged examples and made a forced choice among the alternatives. The result is not strictly comparable to results on the larger tagset, but since only 5 out of 54K test examples are affected, the difference is negligible. This yielded a multiclass problem with 648K examples and 39 classes. We constructed a separate classifier for unknown words, using AdaBoost.MH. We used hapax legomena (words appearing once) from our training sample to train it. The error rate on unknown words was 19.1%. The overall test error rate was 3.59%, intermediate between the error rates in the two previous experiments.

Experiment 4. One obvious way of reducing the training data would be to train a separate classifier for each word. However, that approach would result in extreme data fragmentation. An alternative is to cut the data in the other direction, and build a separate discriminator for each part of speech, and

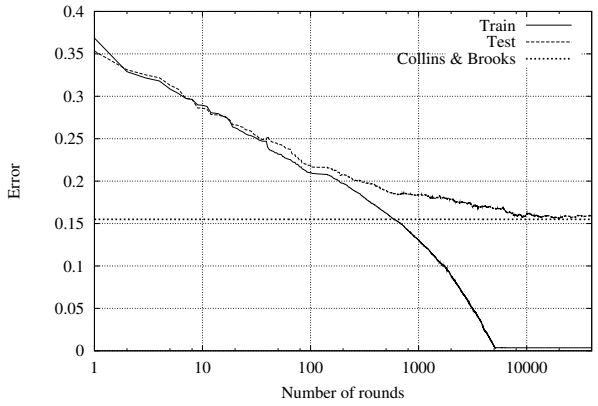

Figure 2: Training and test error as a function of the number of rounds of boosting for the PP-attachment problem.

combine them by choosing the part of speech whose discriminator predicts 'Yes' with the most confidence (or 'No' with the least confidence). We took this approach—algorithm AdaBoost.MI—in experiment 4. To choose the appropriate number of rounds for each discriminator, we did an initial run, and chose the point at which error on the development sample flattened out. To handle unknown words, we used the same unknown-word classifier as in experiment 3.

The result was the best for any of our experiments: a test error rate of 3.28%, slightly better than experiment 2. The 3.28% error rate is not significantly different (at $p = 0.05$) from the error rate of the best-known single tagger, Ratnaparkhi's Maxent tagger, which achieves 3.11% error on our data.

Our results are not as good as those achieved by Brill and Wu's voting scheme. The experiments we describe here use very simple features, like those used in the Maxent or transformation-based taggers; hence the results are not comparable to the multiple-tagger voting scheme. We are optimistic that boosting would do well with tagger predictions as input features, but those experiments remain to be done.

Table 2 breaks out the error sources for experiment 4. Table 3 sums up the results of all four experiments.

Experiment 5 (Sequential model). To this point, tagging decisions are made based on local context only. One would expect performance to improve if we consider a Viterbi-style optimization to choose a globally best sequence of labels. Using decision sequences also permits one to use true tags, rather than most-frequent tags, on context tokens. We did a fixed 500 rounds of boosting, testing against the development sample. Surprisingly, the sequential model performed much less well than the local-decision models. The results are summarized in Table 4.

4 Prepositional phrase attachment

In this section, we discuss the use of boosting for prepositional phrase (PP) attachment. The cases of PP-attachment that we address define a binary classification problem. For example, the sentence *Congress accused the president of peccadillos* is classified according to the attachment site of the prepositional phrase:

$$\begin{array}{l}\text{attachment to N:}\\ \quad\text{accused [the president of peccadillos]}\\ \text{attachment to V:}\\ \quad\text{accused [the president] [of peccadillos]}\end{array} \tag{4}$$

The UPenn Treebank-II Parsed Wall Street Journal corpus includes PP-attachment information, and PP-attachment classifiers based on this data have been previously described in Ratnaparkhi, Reynar, Roukos (1994), Brill and Resnik (1994), and Collins and Brooks (1995). We consider how to apply boosting to this classification task.

We used the same training and test data as Collins and Brooks (1995). The instances of PP-attachment considered are those involving a verb immediately followed by a simple noun phrase (the direct object) and a prepositional phrase (whose attachment is at issue). Each PP-attachment example is represented by its value for four attributes: the main verb (V), the head word of the direct object (N_1), the preposition (P), and the head word of the object of the preposition (N_2). For instance, in example 4 above, $V =$ *accused*, $N_1 =$ *president*, $P =$ *of*, and $N_2 =$ *peccadillos*. Examples have binary labels: positive represents attachment to noun, and negative represents attachment to verb. The training set comprises $20,801$ examples and the test set contains $3,097$ examples; there is also a separate development set of $4,039$ examples.

The weak hypotheses we used correspond to "attribute=value" predicates and conjunctions thereof. That is, there are 16 predicates that are considered for each example. For example 4, three of these 16 predicates are $(V =$ *accused* $\wedge\ N_1 =$ *president* $\wedge\ N_2 =$ *peccadillos*$)$, $(P =$ *with*$)$, and $(V =$ *accused* $\wedge\ P =$ *of*$)$. As described in section 2.1, a weak hypothesis produces one of two real-valued predictions p_0, p_1, depending on the value of

	errors	percent
Local decisions, LT/RT = most-frequent tag	1489/52,087	3.18
Local decisions, LT/RT = true tag	1418/52,087	3.04
Sequential decisions	2083/52,087	4.00

Table 4: Performance of the sequential model on the development sample.

Round	Test	Prediction
1	$(P = \textit{of})$	$+2.393$
2	$(P = \textit{to})$	-0.729
3	$(N_2 = \text{NUMBER})$	-0.772
4	$(N_1 = \textit{it})$	-2.273
5	$(P = \textit{at})$	-0.669

Table 5: The first five weak hypotheses chosen for the PP-attachment classifier.

its predicate. We found that little information was conveyed by knowing that a predicate is false. We therefore forced each weak hypothesis to abstain if its predicate is not satisfied—that is, we set p_0 to 0 for all weak hypotheses.

Two free parameters in boosting are the number of rounds T and the smoothing parameter ε for the confidence values (see Eq. (3)). Although there are theoretical analyses of the number of rounds needed for boosting (Freund and Schapire, 1997; Schapire et al., 1997) and for smoothing (Schapire and Singer, 1998), these tend not to give practical answers. We therefore used the development sample to set these parameters, and chose $T = 20,000$ and $\varepsilon = 0.001$.

On each round of boosting, we consider every predicate relevant to any example, and choose the one that minimizes Z as given by Eq. (2). In Table 5 we list the weak hypotheses chosen on the first five rounds of boosting, together with their assigned confidence p_1. Recall that a positive value means that noun attachment is predicted. Note that all the weak hypotheses chosen on the first rounds test the value of a single attribute: boosting starts with general tendencies and moves toward less widely applicable but higher-precision tests as it proceeds. In $20,000$ rounds of boosting, single-attribute tests were chosen 4,615 times, two-attribute tests were chosen 4,146 times, three-attribute tests were chosen 2,779 times, and four-attribute tests were chosen 8,460 times. It is possible for the same predicate to be chosen in multiple rounds; in fact, predicates were chosen about twice on average. The final hypothesis considers $9,677$ distinct predicates. We can define the total weight of a predicate to be the sum of p_1's over the rounds in which it is chosen; this represents how big a vote the predicate has on examples it applies to. We expect more-specific hypotheses to have more weight—otherwise they would not be able to overrule more-general hypotheses, and there would be no point in having them. This is confirmed by examining the predicates with the greatest weight (in absolute value) after $20,000$ rounds of boosting, as shown in Table 6.

After $20,000$ rounds of boosting the test error was down to $14.6 \pm 0.6\%$. This is indistinguishable from the best known results for this problem, namely, $14.5 \pm 0.6\%$, reported by Collins and Brook on exactly the same data. In Figure 2, we show the training and test error as a function of the number of rounds of boosting. The boosted classifier has the advantage of being much more compact than the large decision list built by Collins and Brooks using a back-off method. We also did not take into account the linguistic knowledge used by Collins and Brooks who, for instance, disallowed tests that ignore the preposition.

Compared to maximum entropy methods (Ratnaparkhi et al., 1994), although the methods share a similar structure, the boosted classifier achieves an error rate which is significantly lower.

5 Using boosting to improve data quality

The importance weights that boosting assigns to training examples are very useful for improving data quality. Mislabelled examples resulting from annotator errors tend to be hard examples to classify correctly; hence they tend to have large weight in the

Test	Prediction
$(V =$ *was*$, N_1 =$ *decision*$, P =$ *of*$, N_2 =$ *People*$)$	$+25.41$
$(V =$ *put*$, N_1 =$ *them*$, P =$ *on*$, N_2 =$ *streets*$)$	-23.08
$(V =$ *making*$, N_1 =$ *it*$, P =$ *in*$, N_2 =$ *terms*$)$	-22.89
$(V =$ *prompted*$, N_1 =$ *speculation*$, P =$ *in*$, N_2 =$ *market*$)$	$+25.76$
$(V =$ *is*$, N_1 =$ *director*$, P =$ *at*$, N_2 =$ *Bank*$)$	$+23.83$

Table 6: The five weak hypotheses with the highest (absolute) weight after $20,000$ rounds.

prev word	tagged word	next word	corpus label	correct label
“	To	be	NN	TO
with	the	Big	JJ	DT
“	the	only	NN	DT
–	and	at	JJ	CC
for	most	of	JJ	JJS
We	have	some	VBN	VBP
–	and	,	JJ	CC
–	a	new	IN	DT
by	A	’s	NNP	DT
<P>	But	in	IN	CC
–	and	what	NN	CC
I	were	out	VB	VBP
n’t	make	the	VBP	VB
have	thought	by	VBD	VBN
will	have	to	VBP	VB
the	first	big	RB	JJ
be	involved	in	JJ	VBN
A	’s	,	NNP	POS
including	as	much	JJ	RB
.	Half	the	DT	PDT
I	were	out	VB	VBP
in	both	gold	CC	(DT)
,	said	to	VBN	
to	one	’s	NN	PRP
to	one	’s	NN	PRP
“	the	only	NN	PRP
to	long-term	,	NN	(RB)
have	called	for	VBD	VBN
have	called	for	VBD	VBN
with	the	Big	JJ	DT
was	his	before	PRP	PRP$
have	thought	by	VBD	VBN
30	%	more	JJ	NN
of	have	and	JJ	

Table 7: Training examples from experiment 4 with greatest weight.

final distribution $D_{T+1}(i)$. If we rank examples by their weight in the final distribution, mislabelled examples tend to cluster near the top.

Table 7 shows the training examples with the greatest weight in tagging experiment 4. All but two represent annotator errors, and one of the two non-errors is a highly unusual construction (“a lot of have and have-not markets”). Table 8 similarly illustrates the highest-weight examples from the PP-attachment data. Many of these are errors, though others are genuinely difficult to judge.

V	N_1	P	N_2	
rose	NUMBER	to	NUMBER	N
dropped	NUMBER	to	NUMBER	N
added	NUMBER	to	NUMBER	N
gained	NUMBER	to	NUMBER	N
gained	NUMBER	to	NUMBER	N
jumped	NUMBER	to	NUMBER	N
reported	earnings	of	million	V
had	sales	of	million	V
lost	NUMBER	to	NUMBER	N
lost	NUMBER	to	NUMBER	N
lost	NUMBER	to	NUMBER	N
earned	million	on	revenue	N
outnumbered	NUMBER	to	NUMBER	V
had	change	in	earnings	V
had	change	in	earnings	V
posted	drop	in	profit	V
yielding	PERCENT	to	assumption	N
posted	loss	for	quarter	V
raise	billion	in	cash	V
is	reporter	in	bureau	V
yield	PERCENT	in	NUMBER	N
yield	PERCENT	in	NUMBER	N
have	impact	on	market	V
posted	drop	in	earnings	V
registered	NUMBER	on	scale	N
auction	million	in	maturity	V
following	decline	in	August	V
reported	earnings	for	quarter	V
signed	agreement	with	Inc.	V
have	impact	on	results	N
report	earnings	for	quarter	N
fell	NUMBER	to	point	N
buy	stake	in	Airlines	V
report	loss	for	quarter	N
make	payments	on	debt	V
took	charge	in	quarter	N
is	writer	in	York	V
earned	million	on	sales	N
earned	million	on	sales	N
reached	agreement	in	principle	V
reached	agreement	in	principle	V
started	venture	with	Co.	N
resolve	disputes	with	company	V
become	shareholder	in	bank	V
reach	agreement	with	regulators	V

Table 8: High-weight examples from the PP-attachment data. The last column gives the label that appears in the corpus.

References

E. Brill and P. Resnik. 1994. A rule-baed appraoch to prepositional phrase attachment disambiguation. In *Proceedings of the fifteenth international conference on computational linguistics (COLING)*.

Eric Brill and Jun Wu. 1998. Classifier combination for improved lexical disambiguation. In *Proceedings of COLING-ACL*.

Eric Brill. 1993. *Transformation-Based Learning*. Ph.D. thesis, Univ. of Pennsylvania.

Michael Collins and James Brooks. 1995. Prepositional phrase attachment through a backed-off model. In *Proceedings of the Third Workshop on Very Large Corpora*.

Yoav Freund 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.

David Magerman. 1995. Statistical decision-tree models for parsing. In *Proc. ACL-95*.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

A. Ratnaparkhi, J. Renyar, and S. Roukos. 1994. A maximum entropy model for prepostional phrase attachement. In *Proceedings of the ARPA Workshop on Human Language Technology*.

Adwait Ratnaparkhi. 1996. A maximum entropy part-of-speech tagger. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*.

Robert E. Schapire 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.

Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. 1997. Boosting the margin: A new explanation for the effectiveness of voting methods. In *Machine Learning: Proceedings of the Fourteenth International Conference*.

Robert E. Schapire. 1997. Using output codes to boost multiclass learning problems. In *Machine Learning: Proceedings of the Fourteenth International Conference*.

Ralph Weischedel et al. 1993. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382.