Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part of Speech Tagging

Eric Brill*
The Johns Hopkins University

Recently, there has been a rebirth of empiricism in the field of natural language processing. Manual encoding of linguistic information is being challenged by automated corpus-based learning as a method of providing a natural language processing system with linguistic knowledge. Although corpus-based approaches have been successful in many different areas of natural language processing, it is often the case that these methods capture the linguistic information they are modelling indirectly in large opaque tables of statistics. This can make it difficult to analyze, understand and improve the ability of these approaches to model underlying linguistic behavior. In this paper, we will describe a simple rule-based approach to automated learning of linguistic knowledge. This approach has been shown for a number of tasks to capture information in a clearer and more direct fashion without a compromise in performance. We present a detailed case study of this learning method applied to part of speech tagging.

1. Introduction

It has recently become clear that automatically extracting linguistic information from a sample text corpus can be an extremely powerful method for overcoming the linguistic knowledge acquisition bottleneck inhibiting the creation of robust and accurate natural language processing systems. A number of part of speech taggers are readily available and widely used, all trained and retrainable on text corpora (Church, 1988; Cutting et al., 1992; Brill, 1992; Weischedel et al., 1993). Endemic structural ambiguity, which can lead to such difficulties as trying to cope with the many thousands of possible parses that a grammar can assign to a sentence, can be greatly reduced by adding empirically derived probabilities to grammar rules (Fujisaki et al., 1989; Sharman, Jelinek, and Mer-

* Department of Computer Science, Baltimore, Md. 21218-2694. Email: brill@cs.jhu.edu.

cer, 1990; Black et al., 1993) and by computing statistical measures of lexical association (Hindle and Rooth, 1993). Word sense disambiguation, a problem which once seemed out of reach for systems without a great deal of hand crafted linguistic and world knowledge, can now in some cases be done with high accuracy when all information is derived automatically from corpora (Brown, Lai, and Mercer, 1991; Yarowsky, 1992; Gale, Church, and Yarowsky, 1992; Bruce and Wiebe, 1994). An effort has recently been undertaken to create automated machine translation systems where the linguistic information needed for translation is extracted automatically from aligned text corpora (Brown et al., 1990). These are just a few of the many recent applications of corpus-based techniques in natural language processing.

Along with great research advances, the infrastructure is in place for this line of research to grow even stronger, with on-line corpora, the grist of the corpus-based natural language processing grindstone, getting bigger and better and becoming more readily available. There are a number of efforts worldwide to manually annotate large corpora with linguistic information, including parts of speech, phrase structure and predicate-argument structure (eg. the Penn Treebank and the British National Corpus (Marcus, Santorini, and Marcinkiewicz, 1993; Leech, Garside, and Bryant, 1994)). A vast amount of on-line text is now available, with much more becoming available in the future. Useful tools, such as large aligned corpora (eg. the aligned Hansards (Gale and Church, 1991)) and semantic word hierarchies (eg. Wordnet (Miller, 1990)), have also recently become available.

Corpus-based methods are often able to succeed while ignoring the true complexities of language, banking on the fact that complex linguistic phenomena can often be indirectly observed through simple epiphenomena. For example, one could accurately assign a part of speech tag to the word **race** in (1-3) without any reference to phrase structure or constituent movement, simply by realizing that it is usually the case that a word that is one or two words to the right of a modal is a verb and not a noun, an exception being when the word is also one word to the right of a determiner.

(1) He **will race/VERB** the car.

(2) He **will** not **race/VERB** the car.

(3) When **will** the **race/NOUN** end?

It is an exciting discovery that simple stochastic n-gram taggers can obtain very high rates of tagging accuracy simply by observing fixed-length word sequences, without recourse to the underlying linguistic structure. However, in order to make progress in corpus-based natural language processing, we must become better aware of just what cues to linguistic structure are really being captured and where the failings are in these approximations to the true underlying phenomena. With many of the current corpus-based approaches to natural language processing, this is a near-impossible task to do. For example, in the part of speech tagging example above, this information about words that follow modals would be hidden deeply in a stochastic n-gram tagger in the thousands or tens of thousands of contextual probabilities ($P(Tag_i|Tag_{i-1}Tag_{i-2})$) and the result of multiplying different combinations of these probabilities together.

Below we describe a new approach to corpus-based natural language processing, called transformation-based error-driven learning. This algorithm has been applied to a number of natural language problems, including part of speech tagging, prepositional phrase attachment disambiguation, and syntactic parsing (Brill, 1992; Brill, 1993a; Brill, 1993b; Brill and Resnik, 1994; Brill, 1994). We have also recently begun exploring the use of this technique for letter to sound generation and for building pronunciation networks for speech recognition. In this approach, the learned linguistic information is represented in a concise and easy to understand form. This property should make transformation-based learning amenable to further exploring linguistic modelling and attempting to discover ways of more tightly coupling the underlying linguistic systems and our approximating models.

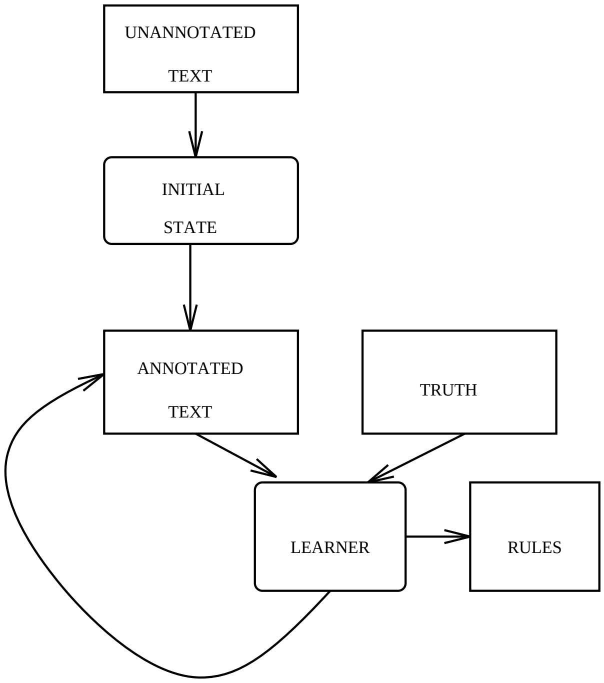


Figure 1

Transformation-Based Error-Driven Learning.

2. Transformation-Based Error-Driven Learning

Figure 1 illustrates how transformation-based error-driven learning works. First, unannotated text is passed through an initial-state annotator. The initial-state annotator can range in complexity from assigning random structure to assigning the output of a sophisticated manually created annotator. In part of speech tagging, various initial state annotators that have been used include: the output of a stochastic n-gram tagger; labelling all words with their most likely tag as indicated in the training corpus; and naively labelling all words as nouns. For syntactic parsing, we have explored initial state annotations ranging from the output of a sophisticated parser to random tree structure with random nonterminal labels.

Once text has been passed through the initial-state annotator, it is then compared to the *truth*. A manually annotated corpus is used as our reference for truth. An ordered list of transformations is learned that can be applied to the output of the initial state annotator to make it better resemble the *truth*. There are two components to a transformation: a rewrite rule and a triggering environment. An example of a rewrite rule for part of speech tagging is:

Change the tag from modal to noun.

and an example of a triggering environment is:

The preceding word is a determiner.

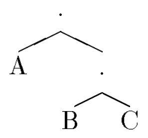
Taken together, the transformation with this rewrite rule and triggering environment when applied to the word **can** would correctly change the mistagged:

The/determiner can/**modal** rusted/verb ./.

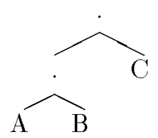
to

The/determiner can/**noun** rusted/verb ./.

An example bracketing rewrite rule is: change the bracketing of a subtree from:



to



where A, B and C can be either teminals or nonterminals. One possible set of triggering environments is any combination of words, part of speech tags, and nonterminal labels within and adjacent to the subtree. Using this rewrite rule and the triggering environment $A = the$, the bracketing:

(the (boy ate))

would become:

((the boy) ate)

In all of the applications we have examined to date, the following greedy search is applied for deriving a list of transformations: at each iteration of learning, the transformation is found whose application results in the *best* score according to the objective function being used; that transformation is then added to the ordered transformation list and the training corpus is updated by applying the learned transformation. Learning continues until no transformation can be found whose application results in an improvement to the annotated corpus. Other more sophisticated search techniques could be used, such as simulated annealing or learning with a look-ahead window, but we have not yet explored these alternatives.

Figure 2 shows an example of learning transformations. In this example, we assume there are only four possible transformations, T1 through T4, and that the objective function is the total number of errors. The unannotated training corpus is processed by the initial state annotator, and this results in an annotated corpus with 5,100 errors, determined by comparing the output of the initial state annotator with the manually derived annotations for this corpus. Next, we apply each of the possible transformations in turn and score the resulting annotated corpus.[1] In this example we see that applying transformation T2 results in the largest reduction of errors, and so T2 is learned as the first transformation. T2 is then applied to the entire corpus, and learning continues. At this stage of learning, transformation T3 results in the largest reduction of error, and so it is learned as the second transformation. After applying the initial state annotator, then T2 and then T3, we see that no further reduction in errors can be obtained from applying any of the transformations, and so learning stops. To annotate fresh text, this text is first

1 In the real implementation, the search is data driven, and therefore all transformations need not be examined.

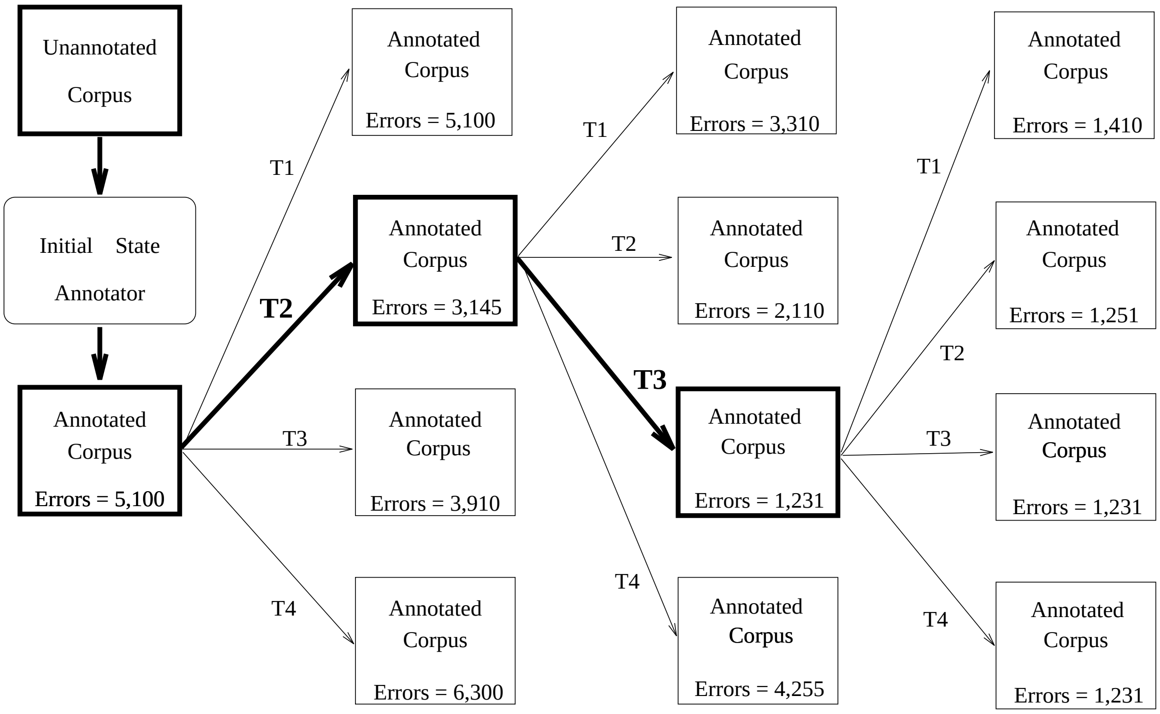


Figure 2

An Example of Transformation-Based Error-Driven Learning.

annotated by the initial state annotator, followed by the application of transformation T2 and then by the application of T3.

To define a specific application of transformation-based learning, one must specify the following:

(1) The initial state annotator.

(2) The space of allowable transformations (rewrite rules and triggering environments)

(3) The objective function for comparing the corpus to the *truth* and choosing a transformation.

In cases where the application of a particular transformation in one environment could affect its application in another environment, two additional parameters must be specified: the order in which transformations are applied to a corpus, and whether a transformation is applied immediately or only after the entire corpus has been examined for triggering environments. For example, take the sequence:

A A A A A A

and the transformation: change the label from A to B if the previous label is A. If the effect of the application of a transformation is not written out until the entire file has been processed for that one transformation, then regardless of the order of processing the output will be: B B B B B B, since the triggering environment of a transformation is always checked before that transformation is applied to any surrounding objects in the corpus. If the effect of a transformation is recorded immediately, then processing the string left to right would result in: A B A B A B, whereas processing right to left would result in: B B B B B B.

3. A Comparison With Decision Trees

The technique employed by the learner is somewhat similar to that used in decision trees (Breiman et al., 1984; Quinlan, 1986; Quinlan and Rivest, 1989). A decision tree is trained on a set of preclassified entities and outputs a set of questions that can be asked about an entity to determine its proper classification. Decision trees are built by finding the question whose resulting partition is the purest,[2] splitting the training data according to that question, and then recursively reapplying this procedure on each resulting subset.

We will first show that the set of classifications that can be provided via decision trees is a proper subset of those that can be provided via transformation lists (an ordered list of transformation-based rules), given the same set of primitive questions. We then give some practical differences between the two learning methods.

2 One possible measure for purity is entropy reduction.

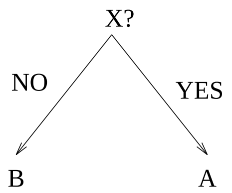
3.1 Decision Trees ⊆ Transformation Lists

We prove here that for a fixed set of primitive queries, any binary decision tree can be converted into a transformation list. Extending the proof beyond binary trees is straightforward.

Proof (by induction):

Base Case:

Given the following primitive decision tree, where the classification is **A** if the answer to the query **X?** is yes, and the classification is **B** if the answer is no:



this tree can be converted into the following transformation list:

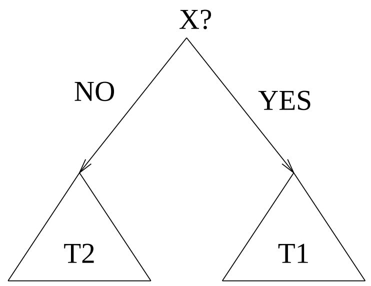
1. Label with S. /* Start State Annotation */

2. If X then S $\rightarrow$ A

3. S $\rightarrow$ B /* Empty Tagging Environment – Always Applies To Entities Currently Labelled With S */

Induction:

Assume that two decision trees T_1 and T_2 have corresponding transformation lists L_1 and L_2. Assume that the arbitrary label names chosen in constructing L_1 are not used in L_2, and that those in L_2 are not used in L_1. Given a new decision tree T_3 constructed from from T_1 and T_2 as follows:



we construct a new transformation list L_3. Assume the first transformation in L_1 is: Label with S', and the first transformation in L_2 is: Label with S". The first three transformations in L_3 will then be:

1. Label with S.
2. If X then S $\rightarrow$ S'
3. S $\rightarrow$ S"

followed by all of the rules in L_1 other than the first rule, followed by all of the rules in L_2 other than the first rule. The resulting transformation list will first label an item as S' if X is true, or as S" is X is false. Since S' is the initial-state label for L_1, the tranformations from L_1 will then be applied if X is true, and likewise the transformations from L_2 will be applied if X is false.

3.2 Decision Trees $\neq$ Transformation Lists

We show here that there exist transformation lists for which no equivalent decision trees exist, for a fixed set of primitive queries. The following classification problem is one example: given a sequence of characters, classify a character based on whether the position index of a character is divisible by 4, querying only using a context of two characters to the left of the character being classified. Assuming transformations are applied left to right on the sequence, the above classification problem can be solved for arbitrary length sequences if the effect of a transformation is written out immediately, or for sequences up to any prespecified length if a transformation is carried out only after all triggering environments in the corpus are checked. We present the proof for the former case. Given the input sequence:

$\underline{\text{A}}$ A A A $\underline{\text{A}}$ A A A $\underline{\text{A}}$ A
0 1 2 3 4 5 6 7 8 9

the underlined characters should be classified as true because their indices are 0, 4, and 8. To see why a decision tree could not perform this classification, regardless of order of classification, note that both the characters and classification of the two characters before both A_3 and A_4 are the same, although these two characters should be classified differently. Below is a transformation list for performing this classification. Once again, we assume transformations are applied left-to-right and that the result of a transformation is written out immediately, so that the result of applying transformation x to character a_i will always be known when applying transformation x to a_{i+1}.

1. Label with S.

   RESULT: A/S A/S A/S A/S A/S A/S A/S A/S A/S A/S A/S

2. If there is no previous character then S $\rightarrow$ F

   RESULT: A/F A/S A/S A/S A/S A/S A/S A/S A/S A/S A/S

3. If the character two to the left is labelled with F then S $\rightarrow$ F

   RESULT: A/F A/S A/F A/S A/F A/S A/F A/S A/F A/S A/F

4. If the character two to the left is labelled with F then F $\rightarrow$ S

   RESULT: A/F A/S A/S A/S A/F A/S A/S A/S A/F A/S A/S

5. F $\rightarrow$ yes

6. S $\rightarrow$ no

   RESULT: A/yes A/no A/no A/no A/yes A/no A/no A/no A/yes A/no A/no

The extra power of transformation lists comes from the fact that intermediate results from the classification of one object are reflected in the current label of that object, thereby making this intermediate information available to be used in classifying other objects. This is not the case for decision trees, where the outcome of questions asked is saved implicitly by the current location within the tree.

3.3 Some Practical Differences Between Decision Trees and Transformation Lists

There are a number of practical differences between transformation-based error-driven learning and learning decision trees. One difference is that when training a decision tree, each time the depth of the tree is increased, the average amount of training material available per node at that new depth is halved (for a binary tree). In transformation-based learning, the entire training corpus is used for finding all transformations, and therefore this method is more resistant to sparse data problems that arise as the depth of the decision tree being learned increases.

Transformations are ordered, with later transformations being dependent upon the outcome of applying earlier transformations. This allows for intermediate results in classifying one object to be available in classifying other objects. For instance, whether the previous word is tagged as *to-infinitival* or *to-preposition* may be a good cue for determining the part of speech of a word.[3] If initially the word *to* is not reliably tagged everywhere in the corpus with its proper tag (or not tagged at all), then this cue will be unreliable. The transformation-based learner will delay positing a transformation triggered by the tag of the word *to* until other transformations have resulted in a more reliable tagging of this word in the corpus. For a decision tree to take advantage of this information, the decision tree path for a word whose outcome is dependent upon the tagging of *to* would have to have the entire decision tree structure for properly classifying each occurrence of *to* built into the classification for that word. If the classification of *to* were dependent upon the classification of yet another word, this would have to be built into the decision tree as well. Unlike decision trees, intermediate classification results in transformation-

3 The original tagged Brown Corpus (Francis and Kucera, 1982) makes this distinction; the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993) does not.

based learning are available and can be used as classification progresses. Even if decision trees are applied to a corpus in a left-to-right fashion, they only are allowed one pass in which to properly classify.

Since a transformation list is a processor and not a classifier, it can readily be used as a postprocessor to any annotation system. In addition to annotating from scratch, rules can be learned to improve the performance of a mature annotation system by using the mature system as the initial-state annotator. This can have the added advantage that the list of transformations learned using a mature annotation system as the start state annotator provide us with a readable description or classification of the errors the mature system makes, thereby aiding in the refinement of that system. Being a processor gives flexibility beyond the classifier-based decision tree. For example, in applying transformation-based learning to parsing, a rule can apply any structural change to a tree. In tagging, a rule such as:

Change the tag of the current word to X and of the previous word to Y if Z holds

can easily be handled in the processor-based system, while it would be difficult to do this in a classification system.

In transformation-based learning, the objective function used in training is the same as that used for evaluation, whenever this is feasible. In a decision-tree, using system accuracy as an objective function for training typically results in poor performance[4] and some measure of node purity, such as entropy reduction, is used instead. The direct correlation between rules and performance improvement in transformation-based learning can make the learned rules more readily interpretable than decision tree rules for increasing population purity.[5]

4 For a discussion on why this is the case, see (Breiman et al., 1984), pages 94-98.
5 For a discussion of other issues regarding these two learning algorithms, see (Ramshaw and Marcus, 1994)

4. Part of Speech Tagging: A Case Study in Transformation-Based Error-Driven Learning

In this section we describe the practical application of transformation-based learning to part of speech tagging.[6] Part of speech tagging is a good application to test the learner for a number of reasons. There are a number of large tagged corpora available, allowing for a variety of experiments to be run. Part of speech tagging is an active area of research, with a great deal of work having been done in this area over the past few years (eg. (Jelinek, 1985; Church, 1988; DeRose, 1988; Hindle, 1989; DeMarcken, 1990; Merialdo, 1994; Brill, 1992; Black et al., 1992; Cutting et al., 1992; Kupiec, 1992; Charniak et al., 1993; Weischedel et al., 1993; Schutze and Singer, 1994)).

Part of speech tagging is also a very practical application, with uses in many areas, including speech recognition and generation, machine translation, parsing, information retrieval and lexicography. Insofar as tagging can be seen as a prototypical problem in lexical ambiguity, advances in part of speech tagging could readily translate to progress in other areas of lexical, and perhaps structural, ambiguity, such as word sense disambiguation and prepositional phrase attachment disambiguation.[7] Also, it is possible to cast a number of other useful problems as part of speech tagging problems, such as letter to sound translation (Huang, Son-Bell, and Baggett, 1994) and building pronunciation networks for speech recognition. Recently, a method has been proposed for using part of speech tagging techniques as a method for parsing with lexicalized grammars (Joshi and Srinivas, 1994).

When automated part of speech tagging was initially explored (Klein and Simmons, 1963; Harris, 1962), people manually engineered rules for tagging, sometimes with the aid of a corpus. As large corpora became available, it became clear that simple Markov-

6 All of the programs described herein are freely available with no restrictions on use or redistribution. For information on obtaining the tagger, contact the author.

7 In (Brill and Resnik, 1994), we describe an approach to prepositional phrase attachment disambiguation that obtains highly competitive performance compared to other corpus-based solutions to this problem. This system was derived in under two hours from the transformation-based part of speech tagger described in this paper.

model based stochastic taggers that were automatically trained could achieve high rates of tagging accuracy (Jelinek, 1985). Markov-model based taggers assign to a sentence the tag sequence that maximizes $Prob(\text{word}|\text{tag}) * Prob(\text{tag}|\text{previous n tags})$. These probabilities can be estimated directly from a manually tagged corpus.[8] These stochastic taggers have a number of advantages over the manually built taggers, including obviating the need for laborious manual rule construction, and possibly capturing useful information that may not have been noticed by the human engineer. However, stochastic taggers have the disadvantage that linguistic information is only captured indirectly, in large tables of statistics. Almost all recent work in developing automatically trained part of speech taggers has been on further exploring Markov-model based tagging (Jelinek, 1985; Church, 1988; DeRose, 1988; DeMarcken, 1990; Merialdo, 1994; Cutting et al., 1992; Kupiec, 1992; Charniak et al., 1993; Weischedel et al., 1993; Schutze and Singer, 1994).

4.1 Transformation-based Error-driven Part of Speech Tagging

Transformation-based part of speech tagging works as follows.[9] The initial state annotator assigns each word its most likely tag as indicated in the training corpus. The method used for initially tagging unknown words will be described in a later section. An ordered list of transformations is then learned to improve tagging accuracy based on contextual cues. These transformations alter the tagging of a word from **X** to **Y** iff either:

1. The word was not seen in the training corpus **OR**
2. The word was seen tagged with **Y** at least once in the training corpus.

In taggers based on Markov models, the lexicon consists of probabilities of the somewhat counterintuitive but proper form $P(WORD|TAG)$. In the transformation-based tagger, the lexicon is simply a list of all tags seen for a word in the training corpus, with

8 One can also estimate these probabilities without a manually tagged corpus, using a hidden Markov model. However, it appears to be the case that directly estimating probabilities from even a very small manually tagged corpus gives better results than training a hidden Markov model on a large untagged corpus (see (Merialdo, 1994)).

9 Earlier versions of this work were reported in (Brill, 1992; Brill, 1994).

one tag labelled as the most likely. Below we show a lexical entry for the word **half** in the transformation-based tagger. A description of the part of speech tags is provided in Appendix A.

half: CD DT JJ <u>NN</u> PDT RB VB

This entry lists the seven tags seen for **half** in the training corpus, with **NN** marked as the most likely. Below are the lexical entries for **half** in a Markov model tagger, extracted from the same corpus:

$$P(half|CD) = 0.000066$$
$$P(half|DT) = 0.000757$$
$$P(half|JJ) = 0.000092$$
$$P(half|NN) = 0.000702$$
$$P(half|PDT) = 0.039945$$
$$P(half|RB) = 0.000443$$
$$P(half|VB) = 0.000027$$

It is difficult to make much sense of these entries in isolation; they have to be viewed in the context of the many contextual probabilities.

First, we will describe a non-lexicalized version of the tagger, where transformation templates do not make reference to specific words. In the nonlexicalized tagger, the transformation templates we use are:

Change tag **a** to tag **b** when:

1. The preceding (following) word is tagged z.
2. The word two before (after) is tagged z.
3. One of the two preceding (following) words is tagged z.

4. One of the three preceding (following) words is tagged z.

5. The preceding word is tagged z and the following word is tagged w.

6. The preceding (following) word is tagged z and the word two before (after) is tagged w.

where a,b,z and w are variables over the set of parts of speech. To learn a transformation, the learner in essence applies every possible transformation,[10] counts the number of tagging errors after that transformation is applied, and chooses that transformation resulting in the greatest error reduction. Learning stops when no transformations can be found whose application reduces errors beyond some prespecified threshold.

In the experiments described below, processing was done left to right. For each transformation application, all triggering environments are first found in the corpus, and then the transformation triggered by each triggering environment is carried out.

The search is data-driven, so only a very small percentage of possible transformations really need be examined. In figure 4.1 we give pseudocode for the learning algorithm in the case where there is only one transformation template: Change the tag from X to Y if the previous tag is Z. In each learning iteration, the entire training corpus is examined once for every pair of tags X and Y, finding the best transformation whose rewrite changes tag X to tag Y. For every word in the corpus whose environment matches the triggering environment, if word has tag X and X is the correct tag, then making this transformation will result in an additional tagging error, and so we increment the number of errors caused when making the transformation given the part of speech tag of the previous word (lines 8 and 9). If X is the current tag and Y is the correct tag, then the transformation will result in one less error, and so we increment the number of improvements caused when making the transformation given the part of speech tag of the previous word (lines 6 and 7).

10 All possible instantiations of transformation templates.

```
1. apply initial state annotator to corpus
2. while transformations can still be found do
3.   for from_tag = tag_1 to tag_n
4.     for to_tag = tag_1 to tag_n
5.       for corpus_position = 1 to corpus_size
6.         if (correct_tag(corpus_position) == to_tag
              && current_tag(corpus_position) == from_tag)
7.                 num_good_transformations(tag(corpus_position -1))++
8.         else if (correct_tag(corpus_position) == from_tag
              && current_tag(corpus_position) == from_tag)
9.                 num_bad_transformations(tag(corpus_position-1 ))++
10.      find max_T (num_good_transformations(T) - num_bad_transformations(T))
11.      if this is the best scoring rule found yet then store as best rule:
           Change tag from from_tag to to_tag if previous tag is T
12.  apply best rule to training corpus
13.  append best rule to ordered list of transformations
```

Figure 3

Pseudocode for learning transformations.

In certain cases, a significant speed-up for training the transformation-based tagger can be obtained by indexing in the corpus where different transformations can and do apply. For a description of a fast index-based training algorithm, see (Ramshaw and Marcus, 1994).

In figure 4 we list the first twenty transformations learned from training on the Penn Treebank Wall Street Journal Corpus (Marcus, Santorini, and Marcinkiewicz, 1993).[11] The first transformation states that a noun should be changed to a verb if the previous tag is TO, as in *to/TO conflict/NN→VB with*. The second transformation fixes a tagging such as: *might/MD vanish/VBP→VB*. The third fixes *might/MD not reply/NN→VB*. The tenth transformation is for the token *'s*, which is a separate token in the Penn Treebank. *'s* is most frequently used as a possessive ending, but after a personal pronoun, it is a verb (John 's, compared to he 's). The transformations changing IN to WDT are for tagging the word *that*, to determine in which environments *that* is being used as a synonym of *which*.

4.2 Lexicalizing the Tagger

In general, no relationships between words have been directly encoded in stochastic n-gram taggers.[12] In the Markov model typically used for stochastic tagging, state transition probabilities ($P(Tag_i|Tag_{i-1}\ldots Tag_{i-n})$) express the likelihood of a tag immediately following n other tags, and emit probabilities ($P(Word_j|Tag_i)$) express the likelihood of a word given a tag. Many useful relationships, such as that between a word and the previous word, or between a tag and the following word, are not directly captured by Markov-model based taggers. The same is true of the non-lexicalized transformation-based tagger, where transformation templates do not make reference to words.

To remedy this problem, we extend the transformation-based tagger by adding contextual transformations that can make reference to words as well as part of speech tags.

11 Version 0.5 of the Penn Treebank was used in all experiments reported in this paper.
12 In (Kupiec, 1992), a limited amount of lexicalization is introduced by having a stochastic tagger with word states for the 100 most frequent words in the corpus.

	Change Tag		
#	From	To	Condition
1	NN	VB	Previous tag is *TO*
2	VBP	VB	One of the previous three tags is *MD*
3	NN	VB	One of the previous two tags is *MD*
4	VB	NN	One of the previous two tags is *DT*
5	VBD	VBN	One of the previous three tags is *VBZ*
6	VBN	VBD	Previous tag is *PRP*
7	VBN	VBD	Previous tag is *NNP*
8	VBD	VBN	Previous tag is *VBD*
9	VBP	VB	Previous tag is *TO*
10	POS	VBZ	Previous tag is *PRP*
11	VB	VBP	Previous tag is *NNS*
12	VBD	VBN	One of previous three tags is *VBP*
13	IN	WDT	One of next two tags is *VB*
14	VBD	VBN	One of previous two tags is *VB*
15	VB	VBP	Previous tag is *PRP*
16	IN	WDT	Next tag is *VBZ*
17	IN	DT	Next tag is *NN*
18	JJ	NNP	Next tag is *NNP*
19	IN	WDT	Next tag is *VBD*
20	JJR	RBR	Next tag is *JJ*

Figure 4

The first 20 non-lexicalized transformations.

The transformation templates we add are:

Change tag **a** to tag **b** when:

1. The preceding (following) word is *w*.
2. The word two before (after) is *w*.
3. One of the two preceding (following) words is *w*.
4. The current word is *w* and the preceding (following) word is *x*.
5. The current word is *w* and the preceding (following) word is tagged *z*.
6. The current word is *w*.
7. The preceding (following) word is *w* and the preceding (following) tag is *t*.
8. The current word is *w*, the preceding (following) word is w_2 and the preceding (following) tag is *t*.

where w and x are variables over all words in the training corpus, and z is a variable over all parts of speech. Below we list two lexicalized transformations that were learned training once again on the Wall Street Journal.

Change the tag:

(12) From **IN** to **RB** if the word two positions to the right is **as**.
(16) From **VBP** to **VB** if one of the previous two words is **n't**.[13]

The Penn Treebank tagging style manual specifies that in the collocation *as* ... *as*, the first *as* is tagged as an adverb and the second is tagged as a preposition. Since *as* is most frequently tagged as a preposition in the training corpus, the initial state tagger

13 In the Penn Treebank, *n't* is treated as a separate token, so *don't* becomes *do/VBP n't/RB*.

*	RB	VBP
*	RB	VB
RB	*	VBP
RB	*	VB

Figure 5

Trigram Tagger Probability Tables.

will mistag the phrase *as tall as* as:

as/**IN** tall/JJ as/IN

The first lexicalized transformation corrects this mistagging. Note that a bigram tagger trained on our training set would not correctly tag the first occurrence of *as*. Although adverbs are more likely than prepositions to follow some verb form tags, the fact that $P(as|IN)$ is much greater than $P(as|RB)$, and $P(JJ|IN)$ is much greater than $P(JJ|RB)$ lead to *as* being incorrectly tagged as a preposition by a stochastic tagger. A trigram tagger will correctly tag this collocation in some instances, due to the fact that $P(IN|RB\ JJ)$ is greater than $P(IN|IN\ JJ)$, but the outcome will be highly dependent upon the context in which this collocation appears.

The second transformation arises from the fact that when a verb appears in a context such as *We do n't eat* or *We did n't usually drink*, the verb is in base form. A stochastic trigram tagger would have to capture this linguistic information indirectly from frequency counts of all trigrams of the form shown in figure 5 (where a star can match any part of speech tag) and from the fact that $P(n't|RB)$ is fairly high.

In (Weischedel et al., 1993), results are given when training and testing a Markov-model based tagger on the Penn Treebank Tagged Wall Street Journal Corpus. They cite results making the closed vocabulary assumption that all possible tags for all words in the test set are known. When training contextual probabilities on 1 million words, an accuracy of 96.7% was achieved. Accuracy dropped to 96.3% when contextual probabilities were trained on 64,000 words. We trained the transformation-based tagger on the same corpus, making the same closed vocabulary assumption.[14] When training contextual rules on 600,000 words, an accuracy of 97.2% was achieved on a separate 150,000 word test set. When the training set was reduced to 64,000 words, accuracy dropped to 96.7%. The transformation-based learner achieved better performance, despite the fact that contextual information was captured in a small number of simple nonstochastic rules, as opposed to 10,000 contextual probabilities that were learned by the stochastic tagger. See table 1. When training on 600,000 words, a total of 447 transformations were learned. However, transformations toward the end of the list contribute very little to accuracy. So only applying the first 200 learned transformations to the test set achieves an accuracy of 97.0%. Applying the first 100 gives an accuracy of 96.8%. To match the 96.7% accuracy achieved by the stochastic tagger when it was trained on one million words, only the first 82 transformations are needed.

To see whether lexicalized transformations were contributing to the transformation-based tagger accuracy rate, we ran the exact same test using the tagger trained using the nonlexical transformation template subset. Accuracy of that tagger was 97.0%. Adding lexicalized transformations resulted in a 6.7% decrease in the error rate. These results are summarized in table 1.[15]

We found it a bit surprising that the addition of lexicalized transformations did not result in a much greater improvement in performance. When transformations are allowed to make reference to words and word pairs, some relevant information is probably missed due to sparse data. We are currently exploring the possibility of incorporating word classes into the rule-based learner in hopes of overcoming this problem. The idea is quite

14 In both (Weischedel et al., 1993) and here, the test set was incorporated into the lexicon, but was not used in learning contextual information. Testing with no unknown words might seem like an unrealistic test. We have done so for three reasons (We show results when unknown words are included later in the paper): (1) to allow for a comparison with previously quoted results, (2) to isolate known word accuracy from unknown word accuracy, and (3) in some systems, such as a closed vocabulary speech recognition system, the assumption that all words are known is valid.

15 The training we did here was slightly suboptimal, in that we used the contextual rules learned with unknown words (described in the next section), and filled in the dictionary, rather than training on a corpus without unknown words.

Method	Training Corpus Size (Words)	# of Rules or Context. Probs.	Acc. (%)
Stochastic	64 K	6,170	96.3
Stochastic	1 Million	10,000	96.7
Rule-Based With Lex. Rules	64 K	215	96.7
Rule-Based With Lex. Rules	600 K	447	97.2
Rule-Based w/o Lex. Rules	600 K	378	97.0

Table 1

Comparison of Tagging Accuracy With No Unknown Words simple. Given any source of word class information, such as WordNet (Miller, 1990), the learner is extended such that a rule is allowed to make reference to parts of speech, words, and word classes, allowing for rules such as *Change the tag from X to Y if the following word belongs to word class Z.* This approach has already been successfully applied to a system for prepositional phrase attachment attachment disambiguation (Brill and Resnik, 1994).

4.3 Tagging Unknown Words

So far, we have not addressed the problem of unknown words. As stated above, the initial state annotator for tagging assigns all words their most likely tag, as indicated in a training corpus. Below we show how a transformation-based approach can be taken for tagging unknown words, by automatically learning cues to predict the most likely tag for words not seen in the training corpus. If the most likely tag for unknown words can be assigned with high accuracy, then the contextual rules can be used to improve accuracy, as described above.

In the transformation-based unknown-word tagger, the initial state annotator naively labels the most likely tag for unknown words as proper noun if capitalized and common noun otherwise.[16]

Below we list the set of allowable transformations.

Change the tag of an unknown word (from X) to Y if:

1. Deleting the prefix (suffix) x, $|x| <= 4$, results in a word (x is any string of length 1 to 4).
2. The first (last) (1,2,3,4) characters of the word are x.
3. Adding the character string x as a prefix (suffix) results in a word ($|x| <= 4$).

16 If we change the tagger to tag all unknown words as common nouns, then a number of rules are learned of the form: **change tag to proper noun if the prefix is "E", "A", "B", etc.**, since the learner is not provided with the concept of upper case in its set of transformation templates.

4. Word W ever appears immediately to the left (right) of the word.

5. Character Z appears in the word.

An unannotated text can be used to check the conditions in all of the above transformation templates. Annotated text is necessary in training to measure the effect of transformations on tagging accuracy. Since the goal is to label each lexical entry for new words as accurately as possible, accuracy is measured on a per type and not a per token basis.

Figure 6 shows the first 20 transformations learned for tagging unknown words in the Wall Street Journal corpus. As an example of how rules can correct errors generated by prior rules, note that applying the first transformation will result in the mistagging of the word *actress*. The 18th learned rule fixes this problem. This rule states: change a tag from **plural common noun** to **singular common noun** if the word has suffix **ss**.

Keep in mind that no specific affixes are prespecified. A transformation can make reference to any string of characters up to a bounded length. So while the first rule specifies the English suffix "s", the rule learner was not constrained from considering such nonsensical rules as: *change a tag to adjective if the word has suffix "xhqr"*. Also, absolutely no English-specific information (such as an affix list) need be prespecified in the learner.[17]

We then ran the following experiment using 1.1 million words of the Penn Treebank Tagged Wall Street Journal Corpus. 950,000 words were used for training and 150,000 words were used for testing. Annotations of the test corpus were not used in any way to train the system. From the 950,000 word training corpus, 350,000 words were used to learn rules for tagging unknown words, and 600,000 words were used to learn contextual rules. 243 rules were learned for tagging unknown words, and 447 contextual tagging

17 This learner has also been applied to tagging Old English. See (Brill, 1993b). Although the transformations are not English-specific, the set of transformation templates would have to be extended to process languages with dramatically different morphology.

	Change Tag		
#	From	To	Condition
1	NN	NNS	Has suffix **-s**
2	NN	CD	Has character **.**
3	NN	JJ	Has character **-**
4	NN	VBN	Has suffix **-ed**
5	NN	VBG	Has suffix **-ing**
6	??	RB	Has suffix **-ly**
7	??	JJ	Adding suffix **-ly** results in a word.
8	NN	CD	The word **$** can appear to the left.
9	NN	JJ	Has suffix **-al**
10	NN	VB	The word **would** can appear to the left.
11	NN	CD	Has character **0**
12	NN	JJ	The word **be** can appear to the left.
13	NNS	JJ	Has suffix **-us**
14	NNS	VBZ	The word **it** can appear to the left.
15	NN	JJ	Has suffix **-ble**
16	NN	JJ	Has suffix **-ic**
17	NN	CD	Has character **1**
18	NNS	NN	Has suffix **-ss**
19	??	JJ	Deleting the prefix **un-** results in a word
20	NN	JJ	Has suffix **-ive**

Figure 6

The first 20 transformations for unknown words.

rules were learned. Unknown word accuracy on the test corpus was 82.2%, and overall tagging accuracy on the test corpus was 96.6%. To our knowledge, this is the highest overall tagging accuracy ever quoted on the Penn Treebank Corpus when making the open vocabulary assumption. Using the tagger without lexicalized rules, an overall accuracy of 96.3% and an unknown word accuracy of 82.0% is obtained. A graph of accuracy as a function of transformation number on the test set for lexicalized rules is shown in figure 7. Before applying any transformations, test set accuracy is 92.4%. So the transformations reduce the error rate by 50% over the baseline. The high baseline accuracy is somewhat misleading, as this includes the tagging of unambiguous words. Baseline accuracy when the words that are unambiguous in our lexicon are not considered is 86.4%. However, it is difficult to compare taggers using this figure, as then the accuracy of the system depends on the particular lexicon used. For instance, in our training set the word *the* was tagged with a number of different tags, and so according to our lexicon *the* is ambiguous. If we instead used a lexicon where *the* is listed unambiguously as a determiner, the baseline accuracy would be 84.6%.

For tagging unknown words, each word is initially assigned a part of speech based on word and word-distribution features. Then, the tag may be changed based on contextual cues, via contextual transformations that are applied to the entire corpus, both known and unknown words. When the contextual rule learner learns transformations, it does so in an attempt to maximize overall tagging accuracy, and not unknown word tagging accuracy. Unknown words account for only a small percentage of the corpus in our experiments, typically two to three percent. Since the distributional behavior of unknown words is quite different from that of known words, and since a transformation that does not increase unknown word tagging accuracy can still be beneficial to overall tagging accuracy, the contextual transformations learned are not optimal in the sense of leading to the highest tagging accuracy on unknown words. Better unknown word accuracy may be possible by training and using two sets of contextual rules, one maximizing known word accuracy and the other maximizing unknown word accuracy, and then applying the

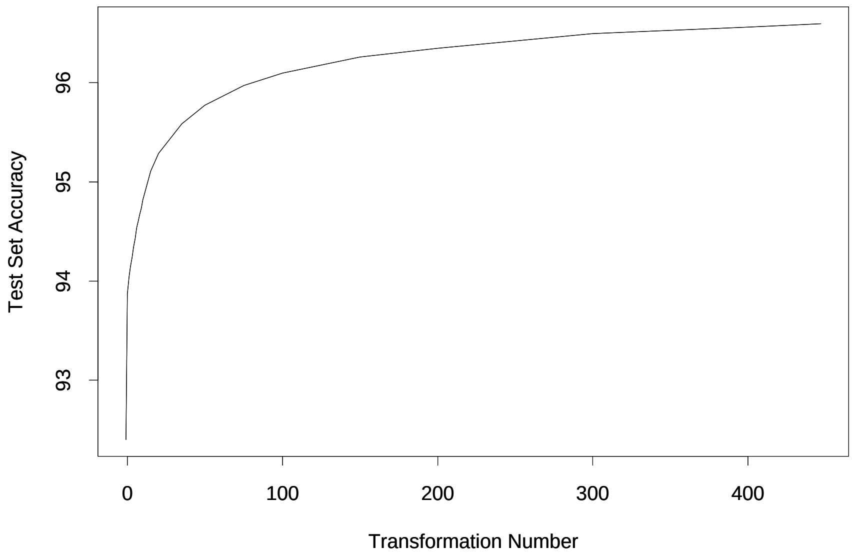


Figure 7

Accuracy vs Transformation Number

Corpus	Accuracy
Penn WSJ	96.6%
Penn Brown	96.3%
Orig Brown	96.5%

Table 2

Tagging Accuracy on Different Corpora

appropriate transformations to a word when tagging, depending upon whether the word appears in the lexicon. We are currently experimenting with this idea.

In (Weischedel et al., 1993), a statistical approach to tagging unknown words is shown. In this approach, a number of suffixes and important features are prespecified. Then, for unknown words:

$$p(W|T) = p(\text{unknown word}|T) * p(\text{Capitalize-feature}|T) * p(suffixes, hyphenation|T)$$

Using this equation for unknown word emit probabilities within the stochastic tagger, an accuracy of 85% was obtained on the Wall Street Journal corpus. This portion of the stochastic model has over 1,000 parameters, with 10^8 possible unique emit probabilities, as opposed to a small number of simple rules that are learned and used in the rule-based approach. In addition, the transformation-based method learns specific cues instead of having them prespecified, allowing for the possibility of uncovering cues not apparent to the human language engineer. We have obtained comparable performance on unknown words, while capturing the information in a much more concise and perspicuous manner, and without prespecifying any information specific to English or to a specific corpus.

In table 2 we show tagging results obtained on a number of different corpora, in each case training on roughly 9.5 x 10^5 words total and testing on a separate test set of 1.5-2 x 10^5 words. Accuracy is consistent across these corpora and tag sets.

In addition to obtaining high rates of accuracy and representing relevant linguistic information in a small set of rules, the part of speech tagger can also be made to run extremely fast. (Roche and Schabes, 1995) shows a method for converting a list of tagging transformations into a deterministic finite state transducer with one state transition taken per word of input, the result being a transformation-based tagger whose tagging speed is about ten times that of the fastest Markov-model tagger.

4.4 K-Best Tags

There are certain circumstances where one is willing to relax the one tag per word requirement in order to increase the probability that the correct tag will be assigned to each word. In (DeMarcken, 1990; Weischedel et al., 1993), k-best tags are assigned within a stochastic tagger by returning all tags within some threshold of probability of being correct for a particular word.

We can modify the transformation-based tagger to return multiple tags for a word by making a simple modification to the contextual transformations described above. The initial-state annotator is the tagging output of the previously described one-best transformation-based tagger. The allowable transformation templates are the same as the contextual transformation templates listed above, but with the rewrite rule *change tag X to tag Y* modified to *add tag X to tag Y* or *add tag X to word W*. Instead of changing the tagging of a word, transformations now add alternative taggings to a word.

When allowing more than one tag per word, there is a trade-off between accuracy and the average number of tags for each word. Ideally, we would like to achieve as large an increase in accuracy with as few extra tags as possible. Therefore, in training we find transformations that maximize the function:

$$\frac{\text{Number of corrected errors}}{\text{Number of additional tags}}$$

In table 3 we present results from first using the one-tag-per-word transformation-based tagger described in the previous section and then applying the k-best tag transformations. These transformations were learned from a separate 240,000 word corpus. As a baseline, we did k-best tagging of a test corpus as follows. Each known word in the test corpus was tagged with all tags seen with that word in the training corpus and the

# of Rules	Accuracy	Avg. # of tags per word
0	96.5	1.00
50	96.9	1.02
100	97.4	1.04
150	97.9	1.10
200	98.4	1.19
250	99.1	1.50

Table 3

Results from k-best tagging.

five most likely unknown word tags were assigned to all words not seen in the training corpus.[18] This resulted in an accuracy of 99.0%, with an average of 2.28 tags per word. The transformation-based tagger obtained the same accuracy with 1.43 tags per word, one third the number of additional tags as the baseline tagger.[19]

5. Conclusions

In this paper, we have described a new transformation-based approach to corpus-based learning. We have given details of how this approach has been applied to part of speech tagging and have demonstrated that the transformation-based approach obtains competitive performance with stochastic taggers on tagging both unknown and known words. The transformation-based tagger captures linguistic information in a small number of simple non-stochastic rules, as opposed to large numbers of lexical and contextual probabilities. This learning approach has also been applied to a number of other tasks, including prepositional phrase attachment disambiguation (Brill and Resnik, 1994), bracketing text (Brill, 1993a) and labeling nonterminal nodes (Brill, 1993c). Recently, we have begun to explore the possibility of extending these techniques to other problems, including learning pronunciation networks for speech recognition and learning mappings between syntactic

18 Thanks to Fred Jelinek and Fernando Pereira for suggesting this baseline experiment.
19 Unfortunately, it is difficult to find results to compare these k-best tag results to. In (DeMarcken, 1990), the test set is included in the training set, and so it is difficult to know how this system would do on fresh text. In (Weischedel et al., 1993), a k-best tag experiment was run on the Wall Street Journal corpus. They quote the average number of tags per word for various threshold settings, but do not provide accuracy results.

and semantic representations.

A. Penn Treebank Part of Speech Tags (Excluding Punctuation)

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential "there"
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PP	Personal pronoun
19.	PP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	"to"
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Acknowledgments

This work was funded in part by NSF grant IRI-9502312. In addition, this work was done in part while the author was in the Spoken Language Systems Group at Massachusetts Institute of Technology under ARPA grant N00014-89-J-1332, and by DARPA/AFOSR grant AFOSR-90-0066 at the University of Pennsylvania. Thanks to Mitch Marcus, Mark Villain, and the anonymous reviewers for many useful comments on earlier drafts of this paper.

References

Black, Ezra, Fred Jelinek, John Lafferty, David Magerman, Robert Mercer, and Salim Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*. Columbus, Ohio.

Black, Ezra, Fred Jelinek, John Lafferty, Robert Mercer, and Salim Roukos. 1992. Decision tree models applied to the labeling of text with parts-of-speech. In *Darpa Workshop on Speech and Natural Language*. Harriman, N.Y.

Breiman, Leo, Jerome Friedman, Richard Olshen, and Charles Stone. 1984. *Classification and regression trees.* Wadsworth and Brooks.

Brill, Eric. 1992. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, Trento, Italy.

Brill, Eric. 1993a. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of the 31st Meeting of the Association of Computational Linguistics*, Columbus, Oh.

Brill, Eric. 1993b. *A Corpus-Based Approach to Language Learning.* Ph.D. thesis, Department of Computer and Information Science, University of Pennsylvania.

Brill, Eric. 1993c. Transformation-based error-driven parsing. In *Proceedings of the Third International Workshop on Parsing Technologies*, Tilburg, The Netherlands.

Brill, Eric. 1994. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, Seattle, Wa.

Brill, Eric and Philip Resnik. 1994. A transformation-based approach to prepositional phrase attachment disambiguation. In *Proceedings of the Fifteenth International Conference on Computational Linguistics (COLING-1994)*, Kyoto, Japan.

Brown, Peter, John Cocke, Stephen Della Pietra, Vincent Della Pietra, Fred Jelinek, John Lafferty, Robert Mercer, and Paul Roossin. 1990. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85.

Brown, Peter, Jennifer Lai, and Robert Mercer. 1991. Word-sense disambiguation using statistical methods. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca.

Bruce, Rebecca and Janyce Wiebe. 1994. Word-sense disambiguation using decomposable models. In *Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics*, Las Cruces, NM.

Charniak, Eugene, Curtis Hendrickson, Neil Jacobson, and Michael Perkowitz. 1993. Equations for part of speech tagging. In *Proceedings of the Conference of the American Association for Artificial Intelligence (AAAI-93)*, Washington DC.

Church, Kenneth. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing, ACL*, Austin, Tx.

Cutting, Doug, Julian Kupiec, Jan Pedersen, and Penelope Sibun. 1992. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, Trento, Italy.

DeMarcken, Carl. 1990. Parsing the LOB corpus. In *Proceedings of the 1990 Conference of the Association for Computational Linguistics*, Pittsburgh, Pa.

DeRose, Stephen. 1988. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14.

Francis, Winthrop Nelson and Henry Kucera. 1982. *Frequency analysis of English usage: Lexicon and grammar.* Houghton Mifflin, Boston.

Fujisaki, Tetsu, Fred Jelinek, John Cocke, and Ezra Black. 1989. Probabilistic parsing method for sentence disambiguation. In *Proceedings of the International Workshop on Parsing Technologies.*

Gale, William and Kenneth Church. 1991. A program for aligning sentences in bilingual corpora. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, Berkeley, Ca.

Gale, William, Kenneth Church, and David Yarowsky. 1992. A method for disambiguating word senses in a large corpus. *Computers and the Humanities.*

Harris, Zellig. 1962. *String Analysis of Language Structure.* Mouton and Co., The Hague.

Hindle, D. and M. Rooth. 1993. Structural ambiguity and lexical relations. *Computational Linguistics*, 19(1):103–120.

Hindle, Donald. 1989. Acquiring disambiguation rules from text. In *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, BC.

Huang, Caroline, Mark Son-Bell, and David Baggett. 1994. Generation of pronunciations from orthographies using transformation-based error-driven learning. In *International Conference on Speech and Language Processing (ICSLP)*, Yokohama, Japan.

Jelinek, Fred. 1985. *Self-Organized Language Modelling for Speech Recognition*. Dordrecht. In Impact of Processing Techniques on Communication, J. Skwirzinski, ed.

Joshi, Aravind and B. Srinivas. 1994. Disambiguation of super parts of speech (or supertags): Almost parsing. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan.

Klein, Sheldon and Robert Simmons. 1963. A computational approach to grammatical coding of English words. *JACM*, 10.

Kupiec, Julian. 1992. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech and Language*, 6.

Leech, Geoffrey, Roger Garside, and Michael Bryant. 1994. Claws4: The tagging of the British National Corpus. In *Proceedings of the 15th International Conference on Computational Linguistics*, Kyoto, Japan.

Marcus, Mitchell, Beatrice Santorini, and Maryann Marcinkiewicz. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2).

Merialdo, Bernard. 1994. Tagging english text with a probabilistic model. *Computational Linguistics*.

Miller, George. 1990. Wordnet: an on-line lexical database. *International Journal of Lexicography*, 3(4).

Quinlan, J. Ross. 1986. Induction of decision trees. *Machine Learning*, 1:81–106.

Quinlan, J. Ross and Ronald Rivest. 1989. Inferring decision trees using the minimum description length principle. *Information and Computation*, 80.

Ramshaw, Lance and Mitchell Marcus. 1994. Exploring the statistical derivation of transformational rule sequences for part-of-speech tagging. In *The Balancing Act: Proceedings of the ACL Workshop on Combining Symbolic and Statistical Approaches to Language*, New Mexico State University.

Roche, Emmanuel and Yves Schabes. 1995. Deterministic part of speech tagging with finite state transducers. *Computational Linguistics*, 21(2).

Schutze, Hinrich and Yoram Singer. 1994. Part of speech tagging using a variable memory Markov model. In *Proceedings of the Association for Computational Linguistics*, Las Cruces, New Mexico.

Sharman, Robert, Fred Jelinek, and Robert Mercer. 1990. Generating a grammar for statistical training. In *Proceedings of the 1990 Darpa Speech and Natural Language Workshop*.

Weischedel, Ralph, Marie Meteer, Richard Schwartz, Lance Ramshaw, and Jeff Palmucci. 1993. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*.

Yarowsky, David. 1992. Word-sense disambiguation using statistical models of Roget's categories trained on large corpora. In *Proceedings of COLING-92*, pages 454–460, Nantes, France, July.