2 Probabilistic Tagging

The TreeTagger has much in common with a conventional ngram tagger [Church, 1988] [Kempe, 1993]. Both of them model the probability of a tagged sequence of words (in the case of a second order Markov model) recursively by:

$$p(w_. w_. ...w_n, t_. t_. ...t_n) := p(t_n|t_{n-\in}t_{n--})\, p(w_n|t_n)p(w_. w_. ...w_{n--}, t_. t_. ...t_{n--})$$ The methods differ, howe

$|t\ \ _{n-}t\ \ _{n-})$ is estimated.

Ngram taggers often estimate the probability using the following formula based on the maximum likelihood estimation (MLE) principle:

$$p(t_n|t_{n-\in}t_{n--}) \quad = \quad \frac{F(t_{n-\in}t_{n--}t_n)}{F(t_{n-\in}t_{n--})} \tag{2}$$

where $F(t_{n-\in}t_{n--}t_n)$ is the number of occurrences of the trigram $t_{n-\in}t_{n--}t_n$ in the corpus and $F(t_{n-\in}t_{n--})$ is the number of occurrences of the bigram $t_{n-\in}t_{n--}$.

This estimation method is problematic since many frequencies are small so that the corresponding probabilities cannot be estimated reliably. Particular difficulties are posed by zero frequencies: it is difficult to decide whether the corresponding trigram is syntactically incorrect (in which case the probability is zero) or just rare (in which case the probability has a small positive value). Another point is that a robust tagger should be able to cope with ungrammaticalities in the input. Otherwise, ungrammaticalities would lead to the assignment of a zero probability to a whole utterance independent of the sequence of tags. This should be avoided.

Therefore, the above formula is often modified by replacing zero probabilities with a small value and renormalizing the probabilities, so that they sum up to 1 (for a comparison of some of the methods, see [Kempe, 1993]). A proper choice of the replacement value is essential for the quality of the tagging result.

3 The TreeTagger

In contrast to an ngram tagger, the TreeTagger estimates transition probabilities with a binary decision tree. Figure 1 shows a sample decision tree[1]. The probability of a given trigram is determined by following the corresponding path through the tree until a leaf is reached. If we look e.g. for the probability of a noun which is preceded by a determiner and an adjective $p(NN|DET, ADJ)$, we must first answer the test at the root node. Since the tag of the previous word is ADJ, we follow the yes-path. The next test ($tag_{-\in} = DET$) is true as well and we end up at a leaf node. Now, we just have to look for the probability of the tag NN in the table which is attached to this node.

3.1 Construction of the Decision Tree

The decision tree is built recursively from a training set of trigrams using a modified version of the ID3-algorithm [Quinlan, 1983]. In each recursion step, a test is created which divides the set of trigram samples in two subsets with maximal distinctness regarding the probability distribution of the third (predicted) tag. The test examines one of the two preceding tags and checks whether it is identical to a tag t. A test has the following form:

$$tag_{-i} = t; \qquad i \in \{1,2\}; \quad t \in T, \tag{3}$$

where T is the tagset.

At each recursion step, all possible tests are compared and the best one yielding most information is attached to the current node of the decision tree. Then this node is expanded recursively on each

---

[1]

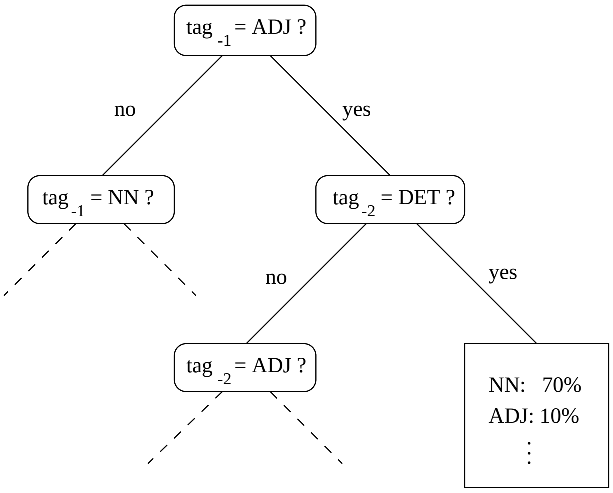

Figure 1: A sample decision tree

of the two subsets of the training set which are defined by the test. The resulting subtrees are attached to the current node as the yes- and no-subtree.

The criterion which is used to compare all possible tests q is the amount of information which is gained about the third tag by making each test. Maximizing the information gain is equivalent to minimizing the average amount of information I_q which is still needed to identify the third tag after the result of the test q is known:

$$I_q = -p(C_{.}|C) \sum_{t \in T} p(t|C_{.})\, log_{.}\, p(t|C_{.}) \;-\; p(C_-|C) \sum_{t \in T} p(t|C_-)\, log_{.}\, p(t|C_-) \tag{4}$$

Here, C is the context which corresponds to the current node and $C_{.}$ (C_-) is equal to C plus the condition that test q succeeds (fails). $p(C_{.}|C)$ $(p(C_-|C))$ is the probability that test q succeeds (fails) and $p(t|C_{.})$ $(p(t|C_-))$ is the probability of the third tag t if the test succeeded (failed). These probabilities are estimated from frequencies[2] with MLE:

$$p(C_{.}|C) = \frac{f(C_{.})}{f(C)} \tag{5}$$

$$p(C_-|C) = \frac{f(C_-)}{f(C)} \tag{6}$$

$$p(t|C_{.}) = \frac{f(t, C_{.})}{f(C_{.})} \tag{7}$$

$$p(t|C_-) = \frac{f(t, C_-)}{f(C_-)} \tag{8}$$

The recursive expansion of the decision tree stops if the next test would generate at least one subset of trigrams whose size is below some predefined threshold, e.g. 2 (i.e. $f(C_{.}) < 2$ or $f(C_-) < 2$). Tag probabilities $p(t|C)$ for the third tag are then estimated from all trigrams which have been passed to this recursion step and they are stored at the current node.

$$p(t|C) := \frac{f(t, C)}{f(C)} \tag{9}$$

---

[2] f,C, ,, t,, ,

3.2 Pruning of the Decision Tree

After the initial version of the decision tree has been built, the tree is pruned. If both subnodes of a node are leaves, and the weighted information gain at the node is below some threshold, the subnodes are removed and the node becomes a leaf itself. The weighted information gain G is defined as:

$$G = f(C)(I_{.} - I_q) \tag{10}$$

$$I_{.} = \sum_{t \in T} p(t|C)\ log_{.}\, p(t|C) \tag{11}$$

where $I_{.}$ is the amount of information which is needed to disambiguate at the current node and I_q, as above, is the amount of information which is still needed after the result of test q is known. This information gain criterion should not be used during the construction of the decision tree, because it is possible that a node fails to meet it, although all its subnodes would. So, this part of the tree would not be constructed if we would use the information gain criterion at first.

As other probabilistic taggers do, the TreeTagger determines the best tag sequence for a given sequence of words with the Viterbi algorithm [Viterbi, 1967].

4 The Lexicon

The lexicon contains the a priori tag probabilities for each word and is similar to the lexicon which was used by [Cutting et al., 1992]. It has three parts: a fullform lexicon, a suffix lexicon and a default entry.

During the lookup of a word in the lexicon of the TreeTagger, the fullform lexicon is searched first. If the word is found there, the corresponding tag probability vector is returned. Otherwise, the uppercase letters of the word are turned to lowercase, and the search in the fullform lexicon is repeated. If it fails again, the suffix lexicon is searched next. If none of the previous steps has been successfull, the default entry of the lexicon is returned.

The fullform lexicon was created from a tagged training corpus (some 2 million words of the Penn Treebank Corpus). The number of occurrences of each word/tag pair was counted and those tags of each word with a relative frequency of less than 1 percent were removed since they were in most cases the result of tagging errors in the original corpus.

The second part of the lexicon, the suffix lexicon, is organised as a tree. Each node of the tree (excepted the root node) is labeled with a character. At the leaf nodes, tag probability vectors are attached. During a lookup, the suffix tree is searched starting a the root node. In each step, the branch which is labeled with the next character from the end of the word suffix is followed.

Assume e.g. we want to look for the word tagging in the suffix lexicon which is shown in fig. 4. We start at the root (labeled #) and follow the branch which leads to the node labeled g. From there, we move to the node labeled n, and finally we end up in the node labeled i. This node is a leaf and the attached tag probability vector (which is not shown in fig. 4) is returned.

The suffix lexicon was automatically built from the training corpus. A suffix tree was constructed from the suffices of length 5 of all words which were annotated with an open class part-of-speech[3] and tag frequencies were counted for all suffices and stored at the corresponding tree nodes. Then an information measure $I(S)$ was calculated for each node of the tree:

$$I(S) = -\sum_{pos} P(pos|S)\ log_{.}\, P(pos|S) \tag{12}$$

Here, S is the suffix which corresponds to the current node and $P(pos|S)$ is the probability of tag *pos* given a word with suffix S.

---

[3]

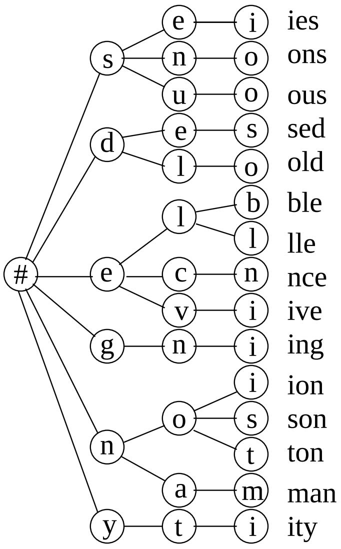

Figure 2: A sample suffix tree of length 3

Using this information measure, the suffix tree has been pruned. For each leaf, the weighted information gain $G(aS)$ was calculated:

$$G(aS) = F(aS)\ (I(S) - I(aS)), \tag{13}$$

where S is the suffix of the parent node, aS is the suffix of the current node and $F(aS)$ is the frequency of suffix aS.

If the information gain at some leaf of the suffix tree is below a given threshold[4], the leaf is removed. The tag frequencies of all deleted subnodes of a parent node are collected at the default node of the parent node. If the default node is the only remaining subnode, it is deleted too. In this case, the parent node becomes a leaf and is also checked whether it is deletable.

To illustrate this process consider the following example, where ess is the suffix of the parent node, less is the suffix of one child node and ness is the suffix of the other child node. Sample tag frequencies of the nodes are given in table 4.

tag	suffix ess	suffix ness	suffix less
JJ	86	1	85
NN	10	2	8
NP	45	45	0
RB	2	0	2
total	143	48	95

Table 1: Sample tag frequencies at a tree node and its two child nodes.

The information measure for the parent node is:

$$I(ess) = -\frac{86}{143} log. \frac{86}{143} - \frac{10}{143} log. \frac{10}{143} - ... \approx 1.32 \tag{14}$$

The corresponding values for the child nodes are 0.39 for ness and 0.56 for less. Now, we can determine the weighted information gain at each of the child nodes. We get:

$$G(ness) = 48(1.32 - 0.39) = 44.64 \tag{15}$$

---

[4]

$$G(less) = 95(1.32 - 0.56) = 72.20 \tag{16}$$

Both values are well above a threshold of 10, and therefore none of them should be deleted.

As explained before, the suffix tree is searched during a lookup along the path, where the nodes are annotated with the letters of the word suffix in reversed order. If a leaf is reached at the end of the path, the corresponding tag probability vector is returned. If no matching subnode can be found at some node on the path, the default node is followed if it exists. If no default node exists, the search in the suffix lexicon fails and the default entry is returned.

The default entry is constructed by subtracting the tag frequencies at all leaves of the pruned suffix tree from the tag frequencies at the root node and normalizing the resulting frequencies. Thereby, relative frequencies are obtained which sum to one.

5 Tests

The performance of the TreeTagger was tested on data from the Penn-Treebank corpus. Some 2 million words were used for training and 100,000 words from a different part of the Penn-Treebank corpus for testing.

The TreeTagger was compared to a trigram tagger [Kempe, 1993] which was trained and tested on the same data. In contrast to the TreeTagger, the trigram tagger does not use a suffix lexicon. Instead, its fullform lexicon was augmented by about 170,000 additional entries which were created from a wordlist with a morphological analyzer.

Two versions of the TreeTagger were tested. In the first version, zero frequencies are replaced by 0.1 before the tag probabilities at the leaves of the decision tree are calculated. In the second version, zero frequencies were replaced by a very small value (10^{--})[5] to see how strong the influence of the choice of this parameter on the tagging accuracy is.

In another test, it was examined how much the tagging accuracy depends on the size of the training corpus. A bigram version and a quatrogram version of the TreeTagger have also been tested here.

Finally, the influence of the pruning threshold on the accuracy of the trigram version and the quatrogram version of the TreeTagger has been tested.

Due to an efficient implementation of the tagger, training with trigrams on 2 million tokens took only about 6 minutes on a SPARC10 workstation and about 10,000 words have been tagged per second.

6 Results

As table 2 shows, the trigram version of the TreeTagger achieved an accuracy which is about 0.3 % better than that of the standard trigram tagger. It is also about 0.6 % better than the bigram version of the TreeTagger. Increasing the context to quatrograms still results in a very small improvement

method	context	accuracy
trigram tagger	trigram	96.06 %
TreeTagger	bigram	95.78 %
TreeTagger (0.1)	trigram	96.34 %
TreeTagger	quatrogram	96.36 %
TreeTagger (10^{--})	trigram	96.32 %

Table 2: Comparison of accuracy

---

[5]

of the accuracy. This shows, that the TreeTagger is able to trim the context effectively, where this is necessary to get reliable estimates.

Changing the replacement value for zero frequencies in the decision tree from a very small value to 0.1 – which was found to be optimal – resulted in a small improvement of the accuracy. So, in contrast to more standard Markov Model taggers, a proper setting of this parameter is not important.

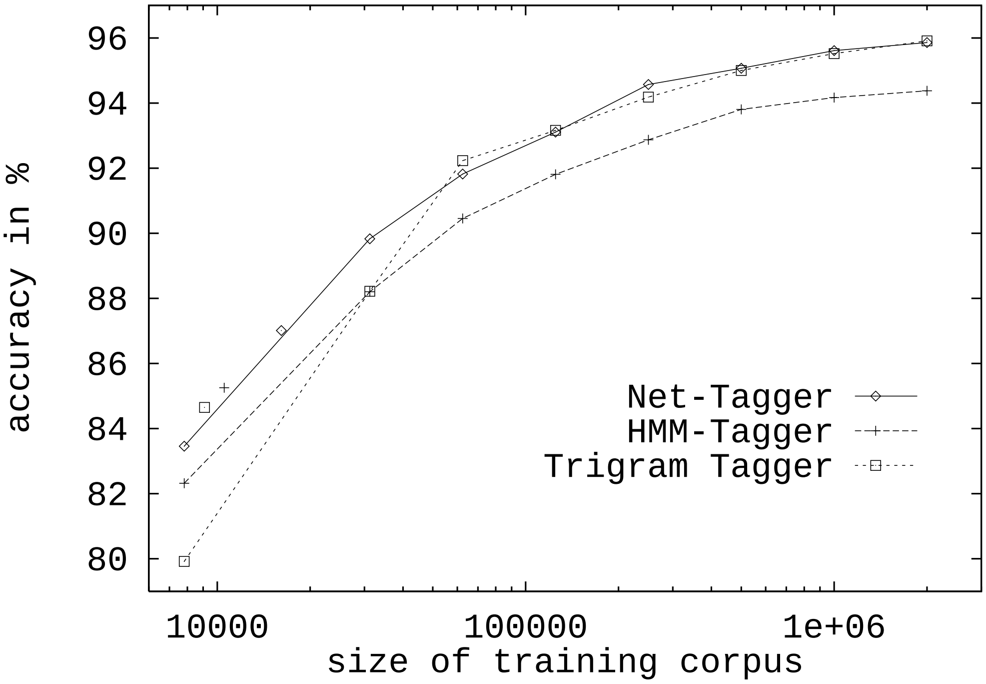

Figure 3: Accuracy for varying sizes of the training corpus.

Figure 3 shows the influence of the size of the training corpus on the tagging quality. Contrary to the trigram tagger, the accuracy of the TreeTagger deteriorates slowly, as the size of the training corpus shrinks. Further, the difference between the bigram version of the TreeTagger and the trigram version is small for very small sizes of the training corpus. This shows that the TreeTagger is robust with regard to the size of the training corpus in contrast to the standard trigram tagger.

Figure 4 shows that the influence of the pruning threshold on the accuracy is negligible (below 0.02 % for the trigram version and below 0.05 % for the quatrogram version). The best results are obtained with thresholds between 40 and 50. Further, the diagrams shows that the quatrogram version of the TreeTagger is consistently better than the trigram version for thresholds above 15.

context	#contexts	#leaves	depth
bigram	47	47	47
trigram	2209	710	62
quatrogram	103823	2251	67

Table 3: Number of ngrams, leaf nodes and the depth of the tree

Table 3 compares the number of possible ngram contexts to the number of leaf nodes and

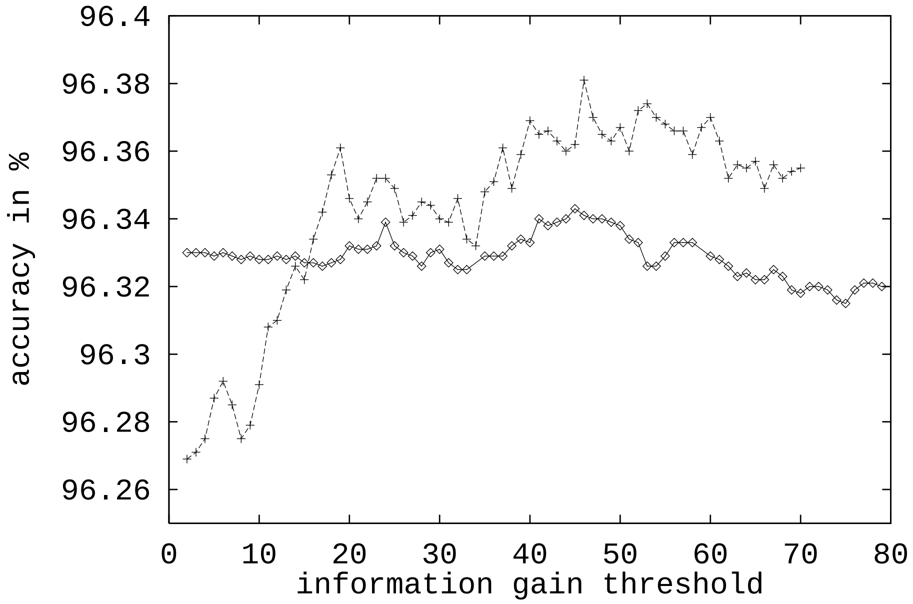

Figure 4: Accuracy for different pruning thresholds

thereby the number of contexts which the TreeTagger distinguishes. In the case of bigrams, all possible contexts are used by the TreeTagger, in the case of trigrams the number of leaves is about a third of the number of possible contexts and in the case of quatrograms only about 2 percent of all possible contexts are distinguished.

7 Summary

A new tagging method, the TreeTagger, has been presented. It differs from other probabilistic taggers in the way the transition probabilities are estimated, namely with a decision tree. A method for the construction of a decision tree was presented and it was demonstrated that the resulting tagger achieves higher accuracy than a standard trigram tagger. Further, it was shown that the TreeTagger is robust with respect to the size of the training corpus. Small training corpora did not result in a sharp degradation of the accuracy, as it was observed for trigram taggers. The TreeTagger trims the size of the context where this is necessary to obtain reliable probability estimates. Hence, it was possible to improve the accuracy of the tagger by using a quatrogram context. Due to an efficient implementation, the tagger is able to tag up to 10,000 tokens per second on a SPARC10-workstation. Thus, the TreeTagger is a fast and high-quality tool for the annotation of corpora with part-of-speech information.

References

[Bahl and Mercer, 1976] Bahl, L. R. and Mercer, R. L. (1976). Part-of-speech assignment by a statistical decision algorithm. IEEE International Symposium on Information Theory, pages 88–89.

[Brill, 1993] Brill, E. (1993). A Corpus-Based Approach To Language Learning. PhD thesis, Department of Computer and Information Science, University of Pennsylvania.

[Church, 1988] Church, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. In Proceedings of the Second Conference on Applied Natural Language Processing, pages 136–143.

[Cutting et al., 1992] Cutting, D., Kupiec, J., Pedersen, J., and Sibun, P. (1992). A practical part-of-speech tagger. In Proceedings of the Third Conference on Applied Natural Language Processing, pages 133–140.

[DeRose, 1988] DeRose, S. J. (1988). Grammatical category disambiguation by statistical optimization. Computational Linguistics, 14(1).

[Federici and Pirrelli, 1994] Federici, S. and Pirrelli, V. (1994). Context-sensitivity and linguistic structure in analogy-based parallel networks. Current Issues in Mathematical Linguistics, pages 353–362.

[Greene and Rubin, 1971] Greene, B. and Rubin, G. (1971). Automatic grammatical tagging of english. Technical report, Department of Linguistics, Brown University, Providence, Rhode Island.

[Kempe, 1993] Kempe, A. (1993). A probabilistic tagger and an analysis of tagging errors. Technical report, Institut für maschinelle Sprachverarbeitung, Universität [Nakamura et al., 1990] Nakamura, M., Maruyama, K.

Neural network approach to word category prediction for english texts. In Karlgren, H., editor, Proceedings of the International Conference on Computational Linguistics, pages 213–218, Helsinki University.

[Quinlan, 1983] Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. In Michalski, R., Carbonell, J., and Mitchell, T., editors, Machine Learning: An artificial intelligence approach, pages 463–482. Morgan Kaufmann, San Mateo, CA.

[Schmid, 1994] Schmid, H. (1994). Part-of-speech tagging with neural networks. In Proceedings of the International Conference on Computational Linguistics, pages 172–176, Kyoto, Japan.

[Viterbi, 1967] Viterbi, A. J. (1967). Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. IEEE Transactions on Information Theory, pages 260–269.