Improvements In Part-of-Speech Tagging With an Application To German*

Helmut Schmid †

IMS-CL
Institut für maschinelle Sprachverarbeitung,
Universität Stuttgart,
Azenbergstr. 12, D-70174 Stuttgart, Germany
email: schmid@ims.uni-stuttgart.de

1 Introduction

Work on part-of-speech tagging has concentrated on English in the past, since a lot of manually tagged training material is available for English and results can be compared to those of other researchers. It was assumed that methods which have been developed for English would work for other languages as well.

There are some special problems to tackle, however, during the development of taggers for other languages. One problem arises from the morphological productivity of languages, as e.g. German, which results in a large number of different word forms which in turn leads to a large number of lexical parameters – at least in the standard Markov Model approach. Another quite general problem is the lack of large, reliably tagged corpora for training purposes. To overcome these problems, methods are needed which achieve high accuracy with small amounts of training data.

A method which has been popular for English is the trigram tagger. It has seldom been used for other languages, however, because the number of parameters becomes too large to be estimated reliably from corpus frequencies. Given a tagset of 50 tags, 125,000 contextual parameters would have to be estimated from perhaps 10,000 to 100,000 tokens of training data. This means that less than one training token is available per parameter on the average. The estimation of lexical probabilities poses similar problems since their number is often even larger.

Unsupervised training on untagged corpora with the Baum-Welch algorithm is an alternative to the direct estimation of Markov Model probabilities from frequency counts. However, Merialdo [Merialdo, 1994] concluded that Baum-Welch reestimation "will generally degrade ... accuracy, except when only a very limited amount of hand-tagged text is available." In his experiments Baum-Welch training did not improve accuracy if more than 5000 training sentences were used for the initialization of parameters.

This paper presents a couple of extensions to a basic Markov Model tagger (called TreeTagger) which improve its accuracy when trained on small corpora. The basic tagger was originally developed for English [Schmid, 1994]. The extensions together reduced error rates on a German test corpus by more than a third.

[*] This is a revised version of a paper which was originally presented at the EACL SIGDAT workshop in Dublin in 1995

[†] This work was supported partially by the Land Baden-Württemberg within the project *Textkorpora und Erschließungswerkzeuge* and partially by the German BMFT within VERBMOBIL.

2 The Basic TreeTagger

The TreeTagger is a Markov Model tagger which makes use of a decision tree to get more reliable estimates for contextual parameters.

2.1 Markov Models

The probability of a sequence of words $w_1 w_2 \ldots w_n$ which is annotated with part of speech (POS) tags may be written as

$$P(w_1 w_2 \ldots w_n, t_1 t_2 \ldots t_n) \tag{1}$$

According to Bayes' theorem, this is equivalent to:

$$P(w_n | w_1 \ldots w_{n-1}, t_1 \ldots t_{n-1} t_n)\ P(t_n | w_1 \ldots w_{n-1}, t_1 \ldots t_{n-1})\ P(w_1 \ldots w_{n-1}, t_1 \ldots t_{n-1}) \tag{2}$$

Using two simplifying assumptions, namely that the probability of a word w_n depends only on its POS t_n and that the probability of the POS depends only on the POSs of the k preceding words, we get the approximation:

$$P(w_n | t_n)\ P(t_n | t_{n-k} \ldots t_{n-1})\ P(w_1 \ldots w_{n-1}, t_1 \ldots t_{n-1}) \tag{3}$$

Recursive application of Bayes' theorem and these simplifying assumptions results in the formula[1]:

$$P(w_1 w_2 \ldots w_n, t_1 t_2 \ldots t_n) = \prod_{i=1}^{n} P(w_i | t_i) P(t_i | t_{i-k} \ldots t_{i-1}) \tag{4}$$

Probabilistic models of this kind where the next state (tag) depends only on the k preceding states (tags) are known as *Markov Models* of k-th order. Mainly first-order Markov models (bigram tagger) and second-order Markov Models (trigram tagger) are used in POS tagging.

The simplifying assumptions above are not fully justified, of course. Neither does a word depend solely on its POS nor is the POS at some position in a sentence solely determined by the POS of the neighboring words. Nevertheless, Markov Modell taggers achieve high accuracy in practice and are very useful due to their robustness and efficiency.

The TreeTagger's scoring function for alternative tag sequences is a slightly modified version of the above formula which results from another application of Bayes' theorem and omission of the factor $P(w_i)$ which is identical for all alternative tag sequences. The new formula simplifies the calculation of lexical probabilities.

$$P(w_1 w_2 \ldots w_n, t_1 t_2 \ldots t_n) = \prod_{i=1}^{n} P(t_i | w_i) / P(t_i)\ P(t_i | t_{i-k} \ldots t_{i-1}) \tag{5}$$

The most probable tag sequence is efficiently calculated by the *Viterbi algorithm.*

2.2 Parameter Estimation

Before a Markov Model tagger can be used to annotate data, it is necessary to estimate the values of its probability parameters. There are essentially two methods for this purpose. One method (see e.g. [Cutting et al., 1992]) estimates the parameters from untagged training data by iterated reestimation with the *Forward-Backward Algorithm.*

[1] We ignore here the fact that $P(t_i | t_{i-k} \ldots t_{i-1})$ is not defined for $i \leq k$. In practice, one can simply define $t_{-k} \ldots t_{-1}$ in some reasonable way, e.g. to mimic the tag sequence at the end of a (not existent) preceding sentence.

The other method needs a training corpus which has been annotated with POS and estimates parameters directly from corpus frequencies (N-gram counts) using *Maximum Likelihood Estimation*:

$$P(t_n|w_n) = \frac{F(t_n, w_n)}{F(w_n)} \tag{6}$$

$$P(t_n|t_{n-k} \dots t_{n-1}) = \frac{F(t_{n-k} \dots t_{n-1}t_n)}{F(t_{n-k} \dots t_{n-1})} \tag{7}$$

The question is how large a training corpus has to be in order to allow reliable estimates from frequencies. From Zipf's law which states that most of the words in a corpus are rare, it is evident that the lexical probabilities $P(w|t)$ can not be estimated reliably for the majority of words. The same holds for the contextual parameters $P(t_n|t_{n-k} \dots t_{n-1})$ if the tagset and/or the context k is large relative to corpus size.

For these reasons it is necessary to either smooth probability estimates or to reduce the number of probability parameters. Smoothing techniques for N-gram counts are discussed e.g. in [Jelinek and Mercer, 1980], [Katz, 1987] and [Church and Gale, 1991].

The TreeTagger follows the other path and reduces the number of contextual parameters via a decision tree method as discussed in the next section. Section 3.1 presents a smoothing method for lexical probabilities.

2.3 Decision Trees for Context Restriction

The selection of a tag based on the sequence of the preceding tags can be seen as a classification problem: Given a set of features (here the preceding tags), what is the most likely class of the item (the tag of the next word)?

One well known method for solving such classification problems is the ID3 algorithm [Quinlan, 1983][2]. It builds a classification tree in which each non-terminal node corresponds to the examination of a feature, while the terminal nodes contain class information. The classification tree is built recursively from a set of training items whose class is known. The ID3 algorithm selects at each step the test which yields maximal information about the class of the training items, splits the training set according to the tested feature and invokes itself recursively to build decision trees for each subset of the training items. The algorithm terminates when the class is unambiguous, i.e. when all training items of the current subtree are of the same class.

A straightforward choice of features for POS prediction are the tags of the k preceding words. Since closer tags provide more information, the algorithm creates in this case a decision tree which examines the preceding tags in sequence until full disambiguation is achieved; the number of examined preceding tags varies depending on the context. Using the tags of the preceding words as features, however, has the disadvantage that each test splits the set of training items in as many subsets as there are tags. While some of the subsets may be quite large, others might be empty and therefore no prediction would be possible. To circumvent this problem and to gain more flexibility, binary features are used. Each binary feature tells whether the last but k word has some tag t. Binary tests in the decision tree are guaranteed to produce non-empty subsets because otherwise the information gain of the test is zero.

Full disambiguation of the tag of the next word on the basis of the preceding words alone is not possible in general. Therefore, unambiguous classifications at the terminal nodes of the decision tree are replaced by probability distributions which reflect the tag distribution of the training set which corresponds to the respective node. To get reliable estimates, these "terminal" training sets have to be sufficiently large. To prevent the algorithm from splitting a training set too far, the TreeTagger applies the heuristic that the *weighted information gain* at a terminal node n has to exceed some threshold Θ_c. The weighted information gain G_n is defined as the product of the frequency F_n of the training items at node n and the difference between the amount of information H_p needed to

[2] A successor of ID3 called C4.5 is also available [Quinlan, 1993].

disambiguate at the parent node p and the amount of information H_n needed to disambiguate at n:

$$\begin{aligned} G_n &= F_n\,(H_n - H_p) && (8) \\ H_x &= -\sum_{tag\ t} p_x(t)\ log\, p_x(t); \quad x = n, p && (9) \end{aligned}$$

The threshold Θ_c is either set to some predefined value or determined by cross evaluation. Fig. 1 shows a sample decision tree.

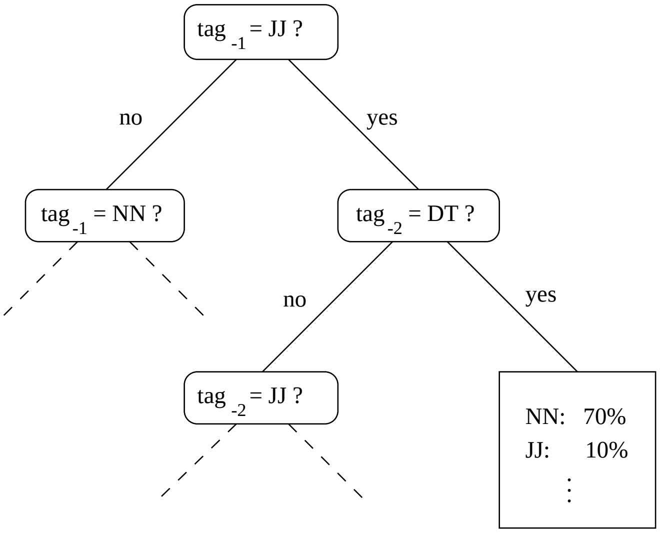

Figure 1: A sample decision tree (partially drawn).

The number of preceding tags which the TreeTagger may examine is defined by the user. Large values were expected to produce the best results. This was confirmed by experiments on English with 2 million words training data from the Penn Treebank corpus [Marcus et al., 1993] where the highest accuracy was achieved with three preceding tags as context. Results for German where less data was available were less clear. Depending on the pruning factor Θ_c, the results were sometimes better with larger contexts and sometimes better with smaller contexts.

2.4 The Suffix Lexicon

When a tagger is processing unrestricted text, it is likely to encounter a number of unknown words even if its lexicon is fairly large. Hence the tagger needs a strategy to deal with unknown words. The simplest strategy is to assign unknown words to each POS tag with equal probability. However, some tags which are called closed class tags (such as determiner, complementizer, preposition) can be safely excluded since the words with these POSs can be listed completely in the lexicon.

A morphological analyzer – if available – can be used to provide more information about unknown words. But since the TreeTagger was designed to work with only a simple full form lexicon and a training corpus, requiring no additional resources, an automatically created *suffix lexicon* is used instead.

The suffix lexicon (see also [Cutting et al., 1992]) assigns tag probabilities to words based on their endings. Construction of the suffix lexicon involves three steps. First a letter tree is built from the suffixes of length five of all open class words in the lexicon and suffix frequencies are counted. The same pruning strategy as for decision trees but with a different threshold Θ_l then removes terminal nodes with unreliable probability estimates. Finally lexical probability estimates are calculated for each terminal node.

Fig. 2 shows a sample suffix tree.

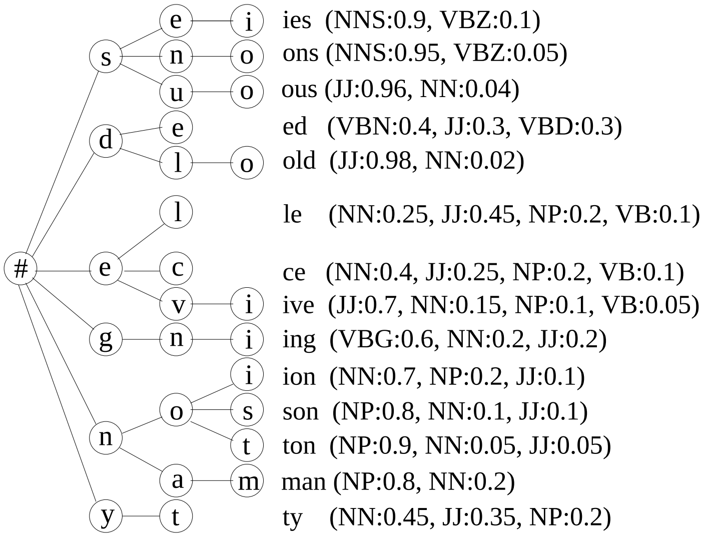
#
s
e
i
n
o
u
o
d
e
l
o
e
l
c
v
i
g
n
i
n
o
i
s
t
a
m
y
t
ies (NNS:0.9, VBZ:0.1)
ons (NNS:0.95, VBZ:0.05)
ous (JJ:0.96, NN:0.04)
ed (VBN:0.4, JJ:0.3, VBD:0.3)
old (JJ:0.98, NN:0.02)
le (NN:0.25, JJ:0.45, NP:0.2, VB:0.1)
ce (NN:0.4, JJ:0.25, NP:0.2, VB:0.1)
ive (JJ:0.7, NN:0.15, NP:0.1, VB:0.05)
ing (VBG:0.6, NN:0.2, JJ:0.2)
ion (NN:0.7, NP:0.2, JJ:0.1)
son (NP:0.8, NN:0.1, JJ:0.1)
ton (NP:0.9, NN:0.05, JJ:0.05)
man (NP:0.8, NN:0.2)
ty (NN:0.45, JJ:0.35, NP:0.2)

Figure 2: A sample suffix tree of maximal length 3.

3 Improvements to the Basic TreeTagger

The tagger described above worked well for English where large training corpora are available. Results for German where only small manually tagged corpora are available were less satisfying. This is not surprising given the fact that only about 50 percent of the word forms in the test text had occurred in the training text. The methods described in the following sections have been developed to improve poor lexical probability estimates obtained from small corpora.

3.1 Smoothing with Equivalence Classes

Grouping words into equivalence classes based on the set of possible tags is used in the Cutting et al. tagger to reduce the number of lexical parameters which have to be reestimated during training. Only one lexical parameter is needed for all words having the same set of possible POSs.

Equivalence classes can also be used for smoothing. If the assumption holds that words with the same set of possible POSs have mostly similar probability distributions, it should be possible to get reasonable probability estimates for rare words from equivalence class probabilities. Tag probabilities for frequent words on the other hand can be based on corpus frequencies. Equation 10 implements a gliding transition from parameter estimation based on equivalence class probabilities to estimation based on corpus frequencies. The equivalence class based lexical probability $P(t\,|\,[w])$ is multiplied by a weight factor α and is added to the frequency $F(t,w)$. The resulting smoothed frequency count $\hat{F}(t,w)$ is used to calculate the lexical probabilities estimates $\hat{P}(t\,|\,w)$.[3]

$$\hat{P}(t\,|\,w) = \frac{\hat{F}(t,w)}{\sum_{t'} \hat{F}(t',w)} \tag{10}$$

$$\hat{F}(t,w) = F(t,w) + \alpha\, P(t\,|\,[w]) \tag{11}$$

[3] The same result is obtained if the word based probabilitity $P(t|w)$ is weighted proportional to word frequency $F(w)$d and added to the equivalence class based probabilitity $P(t|w)$ and renormalized.

3.2 Prefix Lexicon

The influence of the beginning of a word on its POS is small for English words, the only important regularity being that capitalized words are proper names.[4] In such languages as German on the other hand there is a stronger influence since German has also inflectional prefixes.

To model this, the suffix lexicon (see section 2.4) was combined with a prefix lexicon which is built in the same way as the suffix lexicon, but from word prefixes. The probability estimates of the suffix lexicon and the prefix lexicon are multiplied and renormalized in order to be combined.

3.3 Information from Automatically Tagged Corpora

Experiments on unsupervised training of taggers ([Cutting et al., 1992]) have shown that relevant information can also be extracted from untagged corpora.

Untagged corpora presumably are most useful for estimating parameters of words which do not occur in the tagged training corpus or words which are not even contained in the lexicon. For the latter type of words, no information about possible POSs is available to the TreeTagger besides that provided by the prefix-suffix lexicon. There is a good chance, however, to find occurrences of such infrequent words in large untagged corpora. Using the tagger itself (after supervised training) to disambiguate the POSs of these occurrences within context, sets of probable POSs for unknown words can be extracted and added to the full form lexicon so that the tagger has more specific entries for these words than the prefix-suffix lexicon entries which are not very restrictive.

Finding missing lexical entries is one way to make use of automatically tagged data. Another possibility which has been investigated is to calculate probability estimates from such data and to combine them with the lexical probabilities in the same way as the equivalence class probabilities have been combined (see section 3.1).

3.4 Sentence-Initial Words

Since words are always capitalized at the beginning of a sentence, it is not sufficient in this case to look up only the capitalized form of the word in the lexicon. There might be an uncapitalized yet otherwise identical word in the lexicon. Hence the other word form has to be looked up as well and if both lookups are successful, the probability vectors are weighted by the relative frequency of the corresponding forms and summed. This is e.g. necessary to prevent the word *New* in the sentence "New ownership can bring a fresh outlook to stodgy companies." from being tagged as a proper name as in *New York City.*

3.5 Simplified Tagging Formula

Sometimes equation 5 is further simplified to equation 12 by omitting the fraction $1/P(t_i)$. This is justified if the a priori probabilities of all tags are about the same.

$$P(w_1 w_2 \dots w_n, t_1 t_2 \dots t_n) = \prod_{i=1}^{n} P(t_i|w_i)\ P(t_i|t_{i-k} \dots t_{i-1}) \qquad (12)$$

This simplified formula has also been used in initial experiments with the TreeTagger. Tests have shown, however, that the exact formula 5 is superior (see section 5).

4 Tests

A small hand tagged German corpus which is part of a larger corpus of the German newspaper *Stuttgarter Zeitung* was available for tests. The corpus was divided in two subcorpora, one for training (~ 20,000 tokens) and one for testing (~ 5,000 tokens).

[4] Words at the beginning of a sentence are excepted, of course.

tagging method	accuracy
suffix lexicon only (1)	96.05 %
(1) + prefix lexicon	96.10 %
(1) + equival. class smoothing	96.52 %
(1) + sentence initial word treatm.	96.46 %
all features (5)	96.98 %
(5) + additional word/tag-pairs (6)	97.04 %
(6) + additional probabilities	$<$ 97.04 %
(5) + standard MM formula	97.53 %

Table 1: Tagging results

A list of word forms was created from the whole newspaper corpus ($\sim$ 36 million tokens) and analyzed by DMOR, a German morphological analyzer [Schiller, 1995]. All successfully analyzed word forms and the corresponding parts of speech were stored in a full form lexicon. All word/tag pairs from the training corpus which were not yet contained in the full form lexicon were added as well yielding some 350,000 entries altogether.

Different tagger versions have been tested with these data. The first version implemented the simplified tagging formula and included a suffix lexicon but none of the extensions described in section 3. A prefix lexicon was added in the second version. The third version extended version 1 by smoothing with equivalence classes. For version 4, the more sophisticated treatment of sentence-initial word forms was added to version 1. Version 5 contains all three extensions.

Version 6 is identical to version 5 apart from the fact that word-tag pairs from an automatically tagged corpus of 1 million words were added to the full form lexicon if missing. 7700 entries have been added in this way. In version 7 lexical probabilities from the same automatically tagged corpus were merged as described in section 3.3. In the last version the standard tagging formula 5 was used instead of the simplified version (cp. section 3.5).

Tagging speed on a Sun SPARCstation 10 computer was about 8,000 tokens per second.

5 Results

Table 5 shows the percentages of correctly tagged tokens for the different tagger versions.

Smoothing with equivalence classes as well as the more sophisticated treatment of sentence-initial word forms improved accuracy by about 0.5 % and 0.4 % respectively. Improvement from adding a prefix lexicon on the other hand was marginal. Information from the beginning of words was therefore less useful for POS prediction than expected. Adding word/tag pairs from an automatically tagged corpus (version 6) also had only marginal effect on tagging accuracy. Merging lexical probabilities from the same corpus (version 7) even decreased accuracy. How much the accuracy decreased depended on the weight of the probabilities from the automatically tagged corpus.

The results with version 7 show that the standard tagging formula is superior to the simplified one presented in section 3.5. Analysis of the tagger output showed that the latter formula tends to prefer frequent tags. This might have been expected from the fact that the two formulas differ by the factor $P(t)$.

The highest accuracy was achieved by version 7 of the tagger with 97.53 %. This is significantly better than the accuracy obtained with the basic version of the tagger (version 1). The reduction in the error rate is 37 %.

20 % of the errors made by the tagger of version 5 resulted from interchanging a finite verb form and a non-finite verb. These errors were caused by a non-local dependency: it is necessary to know whether an auxiliary precedes in the same clause in order to tell whether a clause-final verb form

which can be either is a finite verb or an infinitive.[5].

The two sentences below illustrate this problem. The verb *eröffnen* in the first sentence was erroneously tagged as a finite form although the modal *wollen* precedes. The verb *halten* in the second sentence is part of a subordinate clause and was incorrectly tagged as a non-finite verb although no modal or auxiliary precedes.

1. Wir **wollen** 1994 die modernste Automobilfabrik in Deutschland **eröffnen**.
   We want to open the most advanced car factory in Germany in 1994.

2. Was Amerikaner augenblicklich für die beste Produktionsmethode **halten**, ...
   What Americans currently consider to be the best production method, ...

This kind of dependency is difficult to learn for an N-gram tagger because it looks only at a limited context. A simple postprocessor can be used to filter out these tagging errors, however. The methods presented in [Brill, 1992] could be used to learn such transformation rules automatically.

18 % of the errors resulted from interchanging a noun and a proper name. Two thirds of the problematic words were either not contained in the full form lexicon (e.g. the proper name *Eberspächer*) or not contained with the correct tag (e.g. the proper name *Trachtenberg* was only listed as a simple noun). 10 % of the problematic words were proper names following an article. The context strongly predicts a noun in these cases.

98.0 % of the test tokens were found in the full form lexicon of version 5. Tagging accuracy on these words was 97.4 %. Accuracy on the remaining 2 % of the tokens which were not contained in the full form lexicon was 78 %.

The average ambiguity of the test tokens was 1.45. This is about the same degree of ambiguity as for the English Penn Treebank data.

Similar tests have been run for English. The English tagger was trained with two preceding tags as context. 2 million tokens from the Penn Treebank corpus [Marcus et al., 1993] were used for training and 100,000 tokens from a different part of the corpus for tests. The lexicon was created from the training corpus. Three different versions of the tagger were tested corresponding to version 1, version 5 and version 7 above. The resulting accuracy was 96.34 % for version 1, 96.44 % for version 5 and 96.81 % for version 7. The improvement in accuracy from version 1 to version 5 is smaller for English than for German. There seems to be less need for smoothing here because the lexical parameters can be estimated quite reliably due to the size of the training corpus.

6 Conclusions

Several extensions to a Markov Model based tagger have been presented which reduced the error rate of the tagger on German data by about 37 %. The best tagger version achieved 97.5 % correctness.

Smoothing lexical probabilities with equivalence class based probabilities as well as a more sophisticated treatment of sentence-initial words have been useful. A limited further improvement was obtained by using a prefix lexicon in addition to a suffix lexicon and by extension of the full form lexicon with word/tag pairs from an automatically tagged corpus. Merging lexical probability estimates from an automatically tagged training corpus, however, had no positive effects. Finally, it was found that the tagger performs better with the standard Markov Model formula which uses $p(w|t)$ as lexical probability than with an alternative formula which uses $p(t|w)$.

The tagger has been implemented in C and is very fast. About 8,000 tokens are tagged per second.

[5] German is a verb-second language. The finite verb is in main clauses (excepted imperative clauses) the second constituent. Any non-finite verbs which form a verb complex together with the finite verb follow at the end of the clause. An arbitrary number of words may intervene between both parts of the verb complex. In subordinate clauses on the other hand the whole verb complex including the finite verb is located at the end of the clause (ignoring shifted constituents which may follow in the so called "Nachfeld" position). A verb at the end of a clause can therefore be either the finite verb of a subordinate clause or a non finite verb in a main clause

References

[Brill, 1992] Brill, E. (1992). A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing*, Trento,Italy.

[Church and Gale, 1991] Church, K. W. and Gale, W. A. (1991). A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams. *Computer Speech and Language*, 5:19–54.

[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.

[Jelinek and Mercer, 1980] Jelinek, F. and Mercer, R. L. (1980). Interpolated estimation of Markov source parameters from sparse data. In *Workshop on Pattern Recognition in Practice*, pages 381–397, Amsterdam.

[Katz, 1987] Katz, S. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on ASSP*, 34(3):400–401.

[Marcus et al., 1993] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

[Merialdo, 1994] Merialdo, B. (1994). Tagging English text with a probabilistic model. *Computational Linguistics*, 20(2):155–171.

[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.

[Quinlan, 1993] Quinlan, J. R. (1993). *C4.5 : Programs for Machine Learning.* Morgan Kaufmann Publishers, San Mateo , CA.

[Schiller, 1995] Schiller, A. (1995). DMOR: Benutzeranleitung. Technical report, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart. (in German).

[Schmid, 1994] Schmid, H. (1994). Probabilistic part-of-speech tagging using decision trees. In *International Conference on New Methods in Language Processing*, pages 44–49, Manchester, UK.