Tagging with Hidden Markov Models Using Ambiguous Tags

Alexis Nasr
LaTTice - Université Paris 7
anasr@linguist.jussieu.fr

Frédéric Béchet
Laboratoire d'Informatique d'Avignon
frederic.bechet@lia.univ-avignon.fr

Alexandra Volanschi
LaTTice - Université Paris 7
avolansk@linguist.jussieu.fr

Abstract

Part of speech taggers based on Hidden Markov Models rely on a series of hypotheses which make certain errors inevitable. The idea developed in this paper consists in allowing a limited, controlled ambiguity in the output of the tagger in order to avoid a number of errors. The ambiguity takes the form of *ambiguous tags* which denote subsets of the tagset. These tags are used when the tagger hesitates between the different components of the ambiguous tags. They are introduced in an existing lexicon and 3-gram database. Their lexical and syntactic counts are computed on the basis of the lexical and syntactic counts of their constituents, using impurity functions. The tagging process itself, based on the Viterbi algorithm, is unchanged. Experiments conducted on the Brown corpus show a recall of 0.982, for an ambiguity rate of 1.233 which is to be compared with a baseline recall of 0.978 for an ambiguity rate of 1.414 using the same ambiguous tags and with a recall of 0.955 corresponding to the one best solution of standard tagging (without ambiguous tags).

1 Introduction

Taggers are commonly used as pre-processors for more sophisticated treatments like full syntactic parsing or chunking. Although taggers achieve high accuracy, they still make some mistakes that quite often impede the following stages. There are at least two solutions to this problem. The first consists in devising more sophisticated taggers either by providing the tagger with more linguistic knowldge or by refining the tagging process, through better probability estimation, for example. The second strategy consists in allowing some ambiguity in the output of the tagger. It is the second solution that was chosen in this paper. We believe that this is an instance of a more general problem in sequential natural language processing chains, in which a module takes as input the output of the preceding module. Since we cannot, in most cases, expect a module to produce only correct solutions, modules should be able to deal with ambiguous input and ambiguous output. In our case, the input is non ambiguous while the output is ambiguous. From this perspective, the quality of the tagger is evaluated by the trade-off it achieves between accuracy and ambiguity.

The introduction of ambiguous tags in the tagger output raises the question of the processing of these ambiguous tags in the post-tagging stages of the application. Leaving some ambiguity in the output of the tagger only makes sense if these other processes can handle it. In the case of a chunker, ambiguous tags can be taken into account through the use of weighted finite state machines, as proposed in (Nasr and Volanschi, 2004). In the case of a syntactic parser, such a device can usually deal with some ambiguity and discard the incorrect elements of an ambiguous tag when they do not lead to a complete analysis of the sentence. The parser itself acts, in a sense, as a tagger since, while parsing the sentence, it chooses the right tag among a set of possible tags for each word. The reason why we still need a tagger and don't let the parser do the job is time and space complexity. Parsers are usually more time and space consuming than taggers and highly ambiguous tags assignments can lead to prohibitive processing time and memory requirements.

The tagger described in this paper is based on the standard Hidden Markov Model architecture (Charniak et al., 1993; Brants, 2000). Such taggers assign to a sequence of words $W = w_1 \ldots w_n$, the part of speech tag sequence $\hat{T} = \hat{t}_1 \ldots \hat{t}_n$ which maximizes the joint probability $P(T, W)$ where T ranges over all possible tag sequences of length n. The probability $P(T, W)$ is itself decomposed into a product of $2n$ probabilities, n *lexical probabilities* $P(w_i|t_i)$ (emission probabilities of the HMM) and n *syn-*

tactic probabilites (transition probabilities of the HMM). Syntactic probabilities model the probability of the occurrence of tag t_i given a *history* which is the knowledge of the h preceding tags $(t_{i-1} \dots t_{i-h})$. Increasing the length of the history increases the predictive power of the tagger but also the number of parameters to estimate and therefore the amount of training data needed. Histories of length 2 constitute a common trade-off for part of speech tagging.

We define an *ambiguous tag* as a tag that denotes a subset of the original tagset. In the remainder of the paper, tags will be represented as subscripted capitals $T : T_1, T_2 \dots$. Ambiguous tags will be noted with multiple subscripts. $T_{1,3,5}$ for example, denotes the set $\{T_1, T_3, T_5\}$. We define the *ambiguity* of an ambiguous tag as the cardinality of the set it denotes. This notion is extended to non ambiguous tags, which can be seen as singletons, their ambiguity is therefore equal to 1.

Ambiguous tags are actually new tags whose lexical and syntactic probability distributions are computed on the basis of lexical and syntactic distributions of their constituents. The lexical and syntactic probability distributions of $T_{i_1,\dots,i_n}$ should be computed in such a way that, when a word in certain context can be tagged as $T_{i_1}, \dots, T_{i_n}$ with probabilities that are close enough, the tagger should choose the ambiguous tag $T_{i_1,\dots,i_n}$.

The idea of changing the tagset in order to improve tagging accuracy has already been tested by several researchers. (Tufiş et al., 2000) reports experiments of POS tagging of Hungarian with a large tagset (about one thousand different tags). In order to reduce data sparseness problems, they devise a reduced tagset which is used for tagging. The same kind of idea is developed in (Brants, 1995). The major difference between these approaches and ours, is that they devise the reduced tagset in such a way that, after tagging, a unique tag of the extended tagset can be recovered for each word. Our perspective is significantly different since we allow unrecoverable ambiguity in the output of the tagger and leave to the other processing stages the task of reducing it. In the HMM based taggers framework, our work bears a certain resemblance with (Brants, 2000) who distinguishes between reliable and unreliable tag assignments using probabilities computed by the tagger. Unreliable tag assignments are those for which the probability is below a given threshold. He shows that taking into account only reliable assignments can significantly improve the accuracy, from 96.6% to 99.4%. In the latter case, only 64.5% of the words are reliably tagged. For the remaining 35.5%, the accuracy is 91.6%. These figures show that taking into account probabilities computed by the tagger discriminates well these two situations. The main difference between his work and ours is that he does not propose a way to deal with unreliable assignments, which we treat using ambiguous tags.

The paper is structured as follows: section 2 describes how the probability distributions of the ambiguous tags are estimated. Section 3 presents an iterative method to automatically discover *good* ambiguous tags as well as an experiment on the Brown corpus. Section 4 concludes the paper.

2 Computing probability distributions for ambiguous tags

Probabilistic models for part of speech taggers are built in two stages. In a first stage, counts are collected from a tagged training corpus while in the second, probabilities are computed on the basis of these counts. Two type of counts are collected: *lexical counts*, noted $C_l(w, T)$ indicating how many times word w has been tagged T in the training corpus and *syntactic counts* $C_s(T_1, T_2, T_3)$ indicating how many times the tag sequence T_1, T_2, T_3 occurred in the training corpus. Lexical counts are stored in a lexicon and syntactic counts in a 3-gram database.

These *real* counts will be used to compute *fictitious* counts for ambiguous tags on the basis of which probability distributions will be estimated. The rationale behind the computation of the counts (lexical as well as syntactic) of an ambiguous tag $T_{1\dots j}$ is that they must reflect the homogeneity of the counts of $\{T_1 \dots T_j\}$. If they are all equal, the count of $T_{1\dots j}$ should be maximal.

Impurity functions (Breiman et al., 1984) perfectly model this behavior[1]: an impurity function Φ is a function defined on the set of all N-*tuples* of numbers $(p_1, \dots, p_N)$ satisfying $\forall j \in [1, \dots, N], p_j \geq 0$ and $\sum_{j=1}^{N} p_j = 1$ with the following properties:

[1] Entropy would be another candidate for such computation. The same experiments have also been conducted using entropy and lead to almost the same results.

- Φ reaches its maximum at the point $(\frac{1}{N}, \ldots, \frac{1}{N})$
- Φ achieves its minimum at the points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots (0, 0, \ldots, 1)$

Given an impurity function Φ, we define the impurity measure of a *N-tuple* of counts $C = (c_1, \ldots, c_N)$ as follows :

$$I(c_1, \ldots, c_N) = \Phi(f_1, \ldots, f_N) \quad (1)$$

where f_i is the relative frequency of c_i in C:

$$f_i = \frac{c_i}{\sum_{k=1}^{N} c_k}$$

The impurity function we have used is the Gini impurity criteria:

$$\Phi(f_1, \ldots, f_N) = \sum_{i \neq j} f_i f_j$$

whose maximal value is equal to $\frac{N-1}{N}$.

The impurity measure will be used to compute both lexical and syntactic fictitious counts as described in the two following sections.

2.1 Lexical counts

Lexical counts for an ambiguous tag $T_{1,\ldots,n}$ are computed using *lexical impurity* $I_l(w, T_{1,\ldots,n})$ which measures the impurity of the *n-tuple* $(C_l(w, T_1), \ldots, C_l(w, T_n))$:

$$I_l(w, T_{1,\ldots,n}) = I(C_l(w, T_1), \ldots, C_l(w, T_n))$$

A high lexical impurity $I_l(w, T_{1,\ldots,n})$ means that w is ambiguous with respect to the different classes $T_1, \ldots, T_n$. It reaches its maximum when w has the same probability to belong to any of them. The lexical count $C_l(w, T_{1,\ldots,n})$ is computed using the following formula:

$$C_l(w, T_{1,\ldots,n}) = I_l(w, T_{1,\ldots,n}) \sum_{i=1}^{n} C_l(w, T_i)$$

This formula is used to update a lexicon, for each lexical entry, the counts of the ambiguous tags are computed and added to the entry. The two entries *daily* and *deals* whose original counts are represented below[2]:

```
daily RB  32 JJ  41
deals NNS 1  VBZ 13
```

are updated to[3]:

```
daily RB  32 JJ  41 JJ_RB   36
deals NNS 1  VBZ 13 NNS_VBZ 2
```

2.2 Syntactic counts

Syntactic counts of the form $C_s(X, Y, T_{1,\ldots,n})$ are computed using *syntactic impurity* $I_s(X, Y, T_{1,\ldots,n})$ which measures the impurity of the *n-tuple* $I(C_s(X, Y, T_1), \ldots, C_s(X, Y, T_n))$:

$$I_s(X, Y, T_{1,\ldots,n}) = I(C_s(X, Y, T_1), \ldots, C_s(X, Y, T_n))$$

A maximum syntactic impurity means that all the tags $T_1, \ldots, T_n$ have the same probability of occurrence after the tag sequence $X\ Y$. If any of them has a probability of occurrence equal to zero after such a tag sequence, the impurity is also equal to zero. The syntactic count $C_s(X, Y, T_{1,\ldots,n})$ is computed using the following formula:

$$C_s(X, Y, T_{1,\ldots,n}) = I_s(X, Y, T_{1,\ldots,n}) \sum_{i=1}^{n} C_s(X, Y, T_i)$$

Such a formula is used to update the 3-gram database in three steps. First, syntactic counts of the form $C_s(X, Y, T_{1,\ldots,n})$ (with X and Y unambiguous) are computed, then syntactic counts of the form $C_s(X, T_{1,\ldots,n}, Y)$ (with X unambiguous and Y possibly ambiguous) and eventually, syntactic counts of the form $C_s(T_{1,\ldots,n}, X, Y)$ (for X and Y possibly ambiguous). The following four real 3-grams:

```
A A A 100    A A B 100
A B A 10     A B B 1000
```

will give rise to following five fictitious ones:

```
A A A_B   100    A A_B A 18
A A_B A_B 31     A A_B B 181
A B A_B   19
```

which will be added to the 3-gram database. Note that the real 3-grams are not modified during this operation.

Once the lexicon and the 3-gram database have been updated, both real and fictitious counts are used to estimate lexical and syntactic probability distribution. These probability distributions constitute the model. The tagging process itself, based on the Viterbi search algorithm, is unchanged.

[2] RB, JJ, NNS and VBZ stand respectively for adverb, adjective, plural noun and verb (3rd person singular, present).

[3] The fictitious counts were rounded to the nearest integer.

2.3 Data sparseness

The introduction of new tags in the tagset increases the number of states in the HMM and therefore the number of parameters to be estimated. It is important to notice that even if the number of parameters increases, the model does not become more sensitive to data sparseness problems than the original model was. The reason is that fictitious counts are computed based on actual counts. The occurrence, in the training corpus, of an event (as the occurrence of a sequence of tags or the occurrence of a word with a given tag) is used for estimating both the probability of the event associated to the simple tag and the probabilities of the events associated with the ambiguous tags which contain the simple tag. For example, the occurrence of the word w with tag T, in the training corpus, will be used to estimate the lexical probability $P(w|T)$ as well as the lexical probabilities $P(w|T')$ for every ambiguous tag T' of which T may be a component.

3 Learning ambiguous tags from errors

Since ambiguous tags are not given *a priori*, candidates can be selected based on the errors made by the tagger. The idea developed in this section consists in learning iteratively ambiguous tags on the basis of the errors made by a tagger. When a word w tagged T_1 in a reference corpus has been wrongly tagged T_2 by the tagger, that means that T_1 and T_2 are lexically and syntactically ambiguous, with respect to w and a given context. Consequently, $T_{1,2}$ is a potential candidate for an ambiguous tag.

The process of discovering ambiguous tags starts with a tagged training corpus whose tagset is called $\mathcal{T}_0$. A standard tagger, $\mathcal{M}_0$, is trained on this corpus. $\mathcal{M}_0$ is used to tag the training corpus. A confusion matrix is then computed and the most frequent error is selected to form an ambiguous tag which is added to $\mathcal{T}_0$ to constitute $\mathcal{T}_1$. $\mathcal{M}_0$ is then updated with the new ambiguous tag to constitue $\mathcal{M}_1$, as described in section 2. The process is iterated : the training corpus is tagged with $\mathcal{M}_i$, the most frequent error is used to constitue $\mathcal{T}_{i+1}$ and a new tagger $\mathcal{M}_{i+1}$ is built, based on $\mathcal{M}_i$. The process continues until the result of the tagging on the development corpus converges or the number of iterations has reached a given threshold.

3.1 Experiments

The model described in section 2 has been tested on the Brown corpus (Francis and Kučera, 1982), tagged with the 45 tags of the Penn treebank tagset (Marcus et al., 1993), which constitute the initial tagset $\mathcal{T}_0$. The corpus has been divided in a training corpus of $961,3$ K words, a development corpus of $118,6$ K words and a test corpus of $115,6$ K words. The development corpus was used to detect the convergence and the final model was evaluated on the test corpus. The iterative tag learning algorithm converged after 50 iterations.

A standard trigram model (without ambiguous tags) $\mathcal{M}_0$ was trained on the training corpus using the CMU-Cambridge statistical language modeling toolkit (Clarkson and Rosenfeld, 1997). Smoothing was done through backoff on bigrams and unigrams using linear discounting (Ney et al., 1994).

The lexical probabilities were estimated on the training corpus. Unknown words (words of the development and test corpus not present in the lexicon) were taken into account by a simple technique: the words of the development corpus not present in the training corpus were used to estimate the lexical counts of unknown words $C_l(\mathtt{UNK}, t)$. During tagging, if a word is unknown, the probability distribution of word `UNK` is used. The development corpus contains 4097 unknown words (3.4% of the corpus) and the test corpus 3991 (3.3%).

3.1.1 Evaluation measures

The result of the tagging process consists in a sequence of ambiguous and non ambiguous tags. This result can no longer be evaluated using accuracy alone (or word error rate), as it is usually the case in part of speech tagging, since the introduction of ambiguous tags allows the tagger to assign multiple tags to a word. This is why two measures have been used to evaluate the output of the tagger with respect to a gold standard: the recall and the ambiguity rate.

Given an output of the tagger $T = t_1 \dots t_n$, where t_i is the tag associated to word i by the tagger, and a gold reference $R = r_1 \dots r_n$ where r_1 is the correct tag for word w_i, the recall of T is computed as follows :

$$REC(T) = \frac{\sum_{i=1}^{n} \delta(r_i \in t_i)}{n}$$

where $\delta(p)$ equals to 1 if predicate p is true and 0 otherwise. A recall of 1 means that for every word occurrence, the correct tag is an element of the tag given by the tagger.

The ambiguity rate of T is computed as follows :

$$AMB(T) = \frac{\sum_{i=1}^{n} AMB(t_i)}{n}$$

where $AMB(t_i)$ is the ambiguity of tag t_i. An ambiguity rate of 1 means that no ambiguous tag has been introduced. The maximum ambiguity rate for the development corpus (when all the possible tags of a word are kept) is equal to 2.4.

3.1.2 Baseline models

The successive models $\mathcal{M}_i$ are based on the different tagsets $\mathcal{T}_i$. Their output is evaluated with the two measures described above. But these figures by themselves are difficult to interpret if we cannot compare them with the output of another tagging process based on the same tagset. The only point of comparision at hand is model $\mathcal{M}_0$ but it is based on tagset $\mathcal{T}_0$, which does not contain ambiguous tags. In order to create such a point of comparison, a baseline model $\mathcal{B}_i$ is built at every iteration. The general idea is to replace in the training corpus, all occurrences of tags that appear as an element of an ambiguous tag of $\mathcal{T}_i$ by the ambiguous tag itself. After the replacement stage, a model $\mathcal{B}_i$ is computed and used to tag the development corpus. The output of the tagging is evaluated using recall and ambiguity rate and can be compared to the output of model $\mathcal{M}_i$.

The replacement stage described above is actually too simplistic and gives rise to very poor baseline models. There are two problems with this approach. The first is that a tag T_i can appear as a member of several ambiguous tags and we must therefore decide which one to choose. The second, is that a word tagged T_i in the reference corpus might be unambiguous, it would therefore be "unfair" to associate to it an ambiguous tag. This is the reason why the replacement step is more elaborate. At iteration i, for each couple (w_j, T_j) of the training corpus, a lookup is done in the lexicon, which gives access to all the possible non ambiguous tags word w_j can have. If there is an ambiguous tag T in $\mathcal{T}_i$ such that all its elements are possible tags of w_j then, couple (w_j, T_j) is replaced with (w_j, T) in the corpus. If several ambiguous tags fulfill this condition, the ambiguous tag which has the highest lexical count for w_j is chosen.

Another simple way to build a baseline would be to produce the n best solutions of the tagger, then take for each word of the input the tags associated to it in the different solutions and make an ambiguous tag out of these tags. This solution was not adopted for two reasons. The first is that this method mixes tags from different solutions of the tagger and can lead to completely incoherent tags sequences. It is difficult to measure the influence of this incoherence on the post-tagging stages of the application and we didn't try to measure it empirically. But the idea of potentially producing solutions which are given very poor probabilities by the model is unappealing. The second reason is that we cannot control anymore which ambiguous tags will be created (although this feature might be desirable in some cases). It will be therefore difficult to compare the result with our models (the tagsets will be different).[4]

3.1.3 Results

The results of the successive models have been plotted in figure 1 and summarized in table 1, which also shows the results on the test corpus. For each iteration i, recall and ambiguity rates of models $\mathcal{M}_i$ and $\mathcal{B}_i$ on the development corpus were computed. The results show, as expected, that recall and ambiguity rate increase with the increase of the number of ambiguous tags added to the tagset. This is true for both models $\mathcal{M}_i$ and $\mathcal{B}_i$. The figure also shows that recall of $\mathcal{B}_i$, for a given i, is generally a bit lower than $\mathcal{M}_i$ while its ambiguity is higher. Figure 2 shows that for the same recall $\mathcal{B}_i$ introduces more ambiguous tags than $\mathcal{M}_i$.

The list of the 20 first ambiguous tags created during the process is represented below :

1	IN_RB	11	IN_WDT_WP
2	DT_IN_WDT_WP	12	VBD_VBN
3	JJ_VBN	13	JJ_NN_NNP_NNS_RB_VBG
4	NN_VB	14	JJ_NN_NNP
5	JJ_NN	15	JJ_NN_NNP_NNS_RB
6	IN_RB_RP	16	JJR_RBR

[4] As a point of comparison we will nevertheless give a few figures here. For low values of n, the n best solutions have better recall for a given value of the ambiguity rate. For instance, the 4 best tagger output yields a recall of 0.9767 for an ambiguity rate of 1.12, while, for the same ambiguity rate, the iterative method obtains a 0.9604 recall. However, the 0.982 recall value which we attained at the end of the iterative ambiguous tag learning procedure, corresponding to an ambiguity rate of 1.23, was also reached by keeping the 7 best solutions of the tagger, with an ambiguity rate of 1.20 (only slightly better than ours).

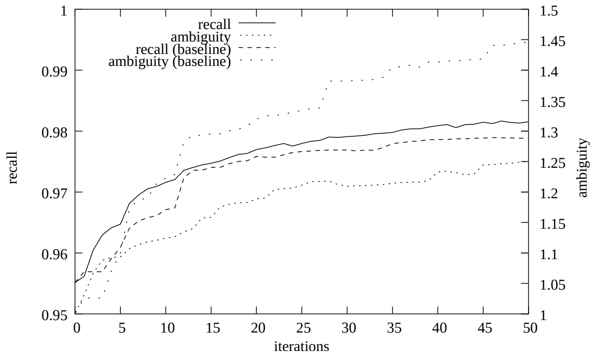

Figure 1: Recall and ambiguity rate of the successive models on development corpus

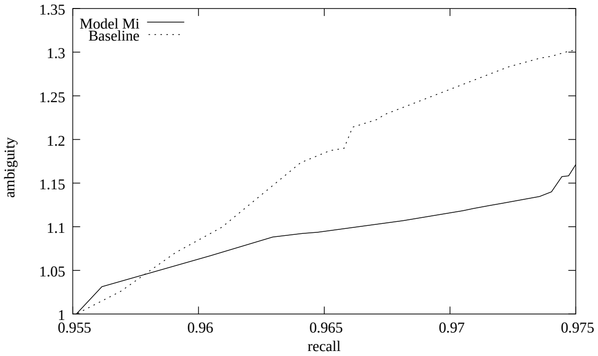

Figure 2: Comparing ambiguity rates for a fixed value of recall

7	NNPS_NNS	17	NN_VBG
8	VB_VBP	18	CD_NN
9	JJ_RB	19	WDT_WP
10	DT_RB	20	JJ_NN_NNP_NNS

Model	DEV		TEST	
	REC	AMB	REC	AMB
$\mathcal{M}_0 = \mathcal{B}_0$	0.955	1	0.955	1
$\mathcal{B}_{40}$	0.978	1.414	0.979	1.418
$\mathcal{M}_{40}$	0.980	1.232	0.982	1.232

Table 1: Results on development and test corpus

3.1.4 Model efficiency

The original idea of our method consists in correcting errors that were made by $\mathcal{M}_0$, through the introduction of ambiguous tags. Ideally, we would like models $\mathcal{M}_i$ with $i > 0$ to introduce an ambiguous tag only where $\mathcal{M}_0$ made a mistake. Unfortunately, it is not always the case. We have classified the use of ambiguous tags into four situations function of their influence on both recall and ambiguity rate as indicated in table 2, where $\mathcal{G}$ stands for the gold standard. In situations 1 and 2 model $\mathcal{M}_0$ made a mistake. In situation 1, the mistake was corrected by the introduction of the ambiguous tag while in situation 2 it was not. In situations 3 and 4, model $\mathcal{M}_0$ did not make a mistake. In situation 3 the introduction of the ambiguous tag did not create a mistake while it did in situation 4.

Situation	$\mathcal{G}$	$\mathcal{M}_0$	$\mathcal{M}_i$	REC	AMB
1	T_1	T_2	$T_{1,2}$	+	+
2	T_3	T_4	$T_{1,2}$	0	+
3	T_1	T_1	$T_{1,2}$	0	+
4	T_3	T_3	$T_{1,2}$	$-$	+

Table 2: Influence of the introduction of an ambiguous tag on recall and ambiguity rates

The frequency of each situation for some of the 20 first ambiguous tags has been reported in table 3. The last column of the table indicates the frequency of the ambiguous tag (number of occurrences of this tag divided by the sum of occurrences of all ambiguous tags). The figures show that ambiguous tags are not very efficient: only a moderate proportion of their occurrences (24% on average) actually corrected an error. While we are very rarely confronted with situation 4 which decreases recall and increases ambiguity (0.5% on average), in the vast majority of cases ambiguous tags simply increase the ambiguity without correcting any mistakes.

Ambiguous tags behave quite differently with respect to the four situations described above. In the best cases (tag 6), 46% of the occurrences corrected an error, and the tag is used one out of ten times the tagger selects an ambiguous tag, as opposed to tag 19 , which corrected errors in 48% of the cases but is not frequently used. The worst configuration is tag 9, which, although not chosen very often, corrects an error in 13% of the occurrences and increases the ambiguity in 85% of its occurrences.

A more detailed evaluation of the basic tagging mistakes has suggested a better adapted and more subtle method of using the ambiguous tags which may at the same time constitute a direction for future work. While the vast majority of mistakes are due to mixing up *word classes*, such as the *-ing* forms used as adjectives, as nouns or as verbs, about one third of the mistakes concern only 25 common *words* such as *that*, *out*, *there*, *on*, *off*, *etc.* Using the ambigu-

Tag	1	2	3	4	freq
1	0.220	0.026	0.746	0.006	0.126
5	0.129	0.014	0.852	0.002	0.165
6	0.461	0.000	0.538	0.000	0.107
9	0.133	0.012	0.850	0.003	0.082
19	0.483	0.064	0.419	0.032	0.012
AVG	0.241	0.029	0.722	0.005	

Table 3: Error analysis of some ambiguous tags

ous tags for these words alone has yielded a recall of 0.965 on the test corpus (25% errors less than model $\mathcal{M}_0$) while keeping the ambiguity rate very low (1.04). With this procedure, 35% of the ambiguous tags occurrences corrected an error made by $\mathcal{M}_0$ and 59% increased the ambiguity. The result can be improved by designing two sets of ambiguous tags: one to be used for this set of words, and one for the word-classes most often mistaken.

4 Conclusions and Future Work

We have presented a method for computing the probability distributions associated to ambiguous tags, denoting subsets of the tagset, in an HMM based part of speech tagger. An iterative method for discovering ambiguous tags, based on the mistakes made by the tagger allowed to reach a recall of 0.982 for an ambiguity rate of 1.232. These figures can be compared to the baseline model which achieves a recall of 0.979 and an ambiguity rate of 1.418 using the same ambiguous tags. An analysis of ambiguous tags showed that they do not always behave in the way expected; some of them introduce a lot of ambiguity without correcting many mistakes.

This work will be developed in two directions. The first one concerns the study of the different behaviour of ambiguous tags which could be influenced by computing differently the fictitious counts of each ambiguous tag, based on its behaviour on a development corpus in order to force or prevent its introduction during tagging. The second direction concerns experiments on supertagging (Bangalore and Joshi, 1999) followed by a parsing stage the tagging stage associates to each word a supertag. The supertags are then combined by the parser to yield a parse of the sentence. Errors of the supertagger (almost one out of 5 words is attributed the wrong supertag) often impede the parsing stage. The idea is therefore to allow some ambiguity during the supertagging stage, leaving to the parser the task of selecting the right supertag using syntactic constraints that are not available to the tagger. Such experiments will constitute one way of testing the viability of our approach.

References

Srinivas Bangalore and Aravind K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2):237–265.

Thorsten Brants. 1995. Tagset reduction without information loss. In *ACL'95*, Cambridge, USA.

Thorsten Brants. 2000. Tnt - a statistical part-of-speech tagger. In *Sixth Applied Natural Language Processing Conference*, Seattle, USA.

L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. 1984. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, California.

Eugene Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz. 1993. Equations for part-of-speech tagging. In *11th National Conference on Artificial Intelligence*, pages 784–789.

Philip Clarkson and Ronald Rosenfeld. 1997. Statistical language modeling using the cmu-cambridge toolkit. In *Eurospeech*.

Nelson Francis and Henry Kučera. 1982. *Frequency Analysis of English Usage: Lexicon and Grammar*. Houghton Mifflin, Boston.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 9(2):313–330, june. Special Issue on Using Large Corpora.

Alexis Nasr and Alexandra Volanschi. 2004. Couplage d'un étiqueteur morpho-syntaxique et d'un analyseur partiel représentés sous la forme d'automates finis pondérés. In *TALN'2004*, pages 329–338, Fez, Morocco.

H. Ney, U. Essen, and R. Kneser. 1994. On structuring probabilistic dependencies in stochastic language modelling. *Computer Speech and Language*, 8:1–38.

Dan Tufiş, Péter Dienes, Csaba Oravecz, and Tamás Váradi. 2000. Principled hidden tagset design for tiered tagging of hungarian. In *LREC*, Athens, Greece.