Unsupervised Training of a Finite-State Sliding-Window Part-of-Speech Tagger[⋆]

Enrique Sánchez-Villamil, Mikel L. Forcada, and Rafael C. Carrasco

Transducens
Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant
E-03071 Alacant

Abstract. A simple, robust sliding-window part-of-speech tagger is presented and a method is given to estimate its parameters from an untagged corpus. Its performance is compared to a standard Baum-Welch-trained hidden-Markov-model part-of-speech tagger. Transformation into a finite-state machine —behaving exactly as the tagger itself— is demonstrated.

1 Introduction

A large fraction (typically 30%, but varying from one language to another) of the words in natural language texts are words that, in isolation, may be assigned more than one morphological analysis and, in particular, more than one part of speech (PoS). The correct resolution of this kind of ambiguity for each occurrence of the word in the text is crucial in many natural language processing applications; for example, in machine translation, the correct equivalent of a word may be very different depending on its part of speech.

This paper presents a sliding-window PoS tagger (SWPoST), that is, a system which assigns the part of speech of a word based on the information provided by a fixed window of words around it. The SWPoST idea is not new; however, we are not aware of any SWPoST which, using reasonable approximations, may easily be trained in an unsupervised manner; that is, avoiding costly manual tagging of a corpus. Furthermore, as with any fixed-window SWPoST, and in contrast with more customary approaches such as hidden Markov models (HMM), the tagger may be implemented exactly as a finite-state machine (a Mealy machine).

The paper is organized as follows: section 2 gives some definitions, describes the notation that will be used throughout the paper, and compares the sliding-window approach to the customary (HMM) approach to part-of-speech tagging; section 3 describes the approximations that allow a SWPoST to be trained in an unsupervised manner and describes the training process itself; section 4 describes how the tagger may be used on new text and how it may be turned into a finite-state tagger; section 5 describes a series of experiments performed to compare

⋆ Work funded by the Spanish Government through grant TIC2003-08681-C02-01.

the performance of a SWPoST to that of a HMM tagger and to explore the size of the resulting finite state taggers (after minimization); and, finally, concluding remarks are given in section 6.

2 Preliminaries

Let $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_{|\Gamma|}\}$ be the *tagset* for the task, that is, the set of PoS tags a word may receive and $W = \{w_1, w_2, \ldots\}$ be the vocabulary of the task. A partition of W is established so that $w_i \equiv w_j$ if and only if both are assigned the same subset of tags. Each of the classes of this partition is usually called an *ambiguity class*. It is usual [1] to refine this partition so that, for high-frequency words, each word class contains just one word whereas, for lower-frequency words, word classes are made to correspond exactly to ambiguity classes (although it is also possible to use one-word classes for all words or to use only ambiguity classes), which allows for improved performance on very frequent ambiguous words while keeping the number of parameters of the tagger under control.

Any such refinement will be denoted as $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{|\Sigma|}\}$ where σ_i are word classes. In this paper, word classes will simply be ambiguity classes, without any refinement. We will call $T : \Sigma \to 2^\Gamma$ the function returning the set $T(\sigma)$ of PoS tags for each word class σ.

The part-of-speech tagging problem may be formulated as follows: given a text $w[1]w[2]\ldots w[L] \in W^+$, each word $w[t]$ is assigned (using a lexicon, a morphological analyser, or a guesser) a word class $\sigma[t] \in \Sigma$ to obtain an *ambiguously tagged* text $\sigma[1]\sigma[2]\ldots\sigma[L] \in \Sigma^+$; the task of the PoS tagger is to obtain a tagged text $\gamma[1]\gamma[2]\ldots\gamma[L] \in \Gamma^+$ (with $\gamma[t] \in T(\sigma[t])$) as correct as possible.

Statistical part-of-speech tagging looks for the *most likely* tagging of an ambiguously tagged text $\sigma[1]\sigma[2]\ldots\sigma[L]$:

$$\gamma^*[1]\ldots\gamma^*[L] = \underset{\gamma[t]\in T(\sigma[t])}{\operatorname{argmax}} P(\gamma[1]\ldots\gamma[L]|\sigma[1]\ldots\sigma[L]) \tag{1}$$

which, using Bayes' formula, becomes equivalent to:

$$\gamma^*[1]\ldots\gamma^*[L] = \underset{\gamma[t]\in T(\sigma[t])}{\operatorname{argmax}} P_S(\gamma[1]\ldots\gamma[L])P_L(\sigma[1]\ldots\sigma[L]|\gamma[1]\ldots\gamma[L])) \tag{2}$$

where $P_S(\gamma[1]\ldots\gamma[L])$ is the probability of a particular tagging (syntactical probability) and $P_L(\sigma[1]\ldots\sigma[L]|\gamma[1]\ldots\gamma[L])$ is the probability of that particular tagging generating the text $\sigma[1]\ldots\sigma[L]$ (lexical probability). In hidden Markov models (HMM) [5], these probabilities are approximated as products; the syntactical probabilities are modeled by a first-order Markov process:

$$P_S(\gamma[1]\gamma[2]\ldots\gamma[L]) = \prod_{t=0}^{t=L} p_S(\gamma[t+1]|\gamma[t]) \tag{3}$$

where $\gamma[0]$ and $\gamma[L+1]$ are fixed delimiting tags (which we will denote as $\gamma_\#$ and will usually correspond to sentence boundaries); lexical probabilities are made independent of context:

$$P_L(\sigma[1]\sigma[2]\ldots\sigma[L]|\gamma[1]\gamma[2]\ldots\gamma[L]) = \prod_{t=1}^{t=L} p_L(\sigma[t]|\gamma[t]). \tag{4}$$

The number of trainable parameters for such a tagger is $(|\Gamma| + |\Sigma|)|\Gamma|$. Tagging (searching for the optimal $\gamma^*[1]\gamma^*[2]\ldots\gamma^*[L]$) is implemented using a very efficient, left-to-right algorithm usually known as Viterbi's algorithm [1, 5]. If conveniently implemented, Viterbi's algorithm can output a partial tagging each time a nonambiguous word is seen, but maintains multiple hypotheses when reading ambiguous words. HMMs may be trained either from tagged text (simply by counting and taking probabilities to be equal to frequencies) or from untagged text, using the well-known expectation-maximization backward-forward Baum-Welch algorithm [5, 1].

In this paper we look at tagging from a completely different perspective. Instead of using the inverted formulation in eq. (2) we approximate the probability in eq. (1) directly as:

$$P(\gamma[1]\gamma[2]\ldots\gamma[L]|\sigma[1]\sigma[2]\ldots\sigma[L]) = \prod_{t=1}^{t=L} p(\gamma[t]|C_{(-)}[t]\sigma[t]C_{(+)}[t]) \tag{5}$$

where

$$C_{(-)}[t] = \sigma[t - N_{(-)}]\sigma[t - N_{(-)} + 1]\cdots\sigma[t-1]$$

is a *left context* of size $N_{(-)}$,

$$C_{(+)}[t] = \sigma[t+1]\sigma[t+2]\cdots\sigma[t+N_{(+)}]$$

is a *right context* of size $N_{(+)}$, and $\sigma[-N_{(-)}+1], \sigma[-N_{(-)}+2], \ldots, \sigma[0]$ and $\sigma[L+1], \sigma[L+2], \ldots \sigma[L+N_{(+)}]$ are all set to a special delimiting word class $\sigma_\#$ such that $T(\sigma_\#) = \{\gamma_\#\}$, e.g., one containing the sentence-boundary marker tag $\gamma_\# \in \Gamma$. This *sliding window* method is local in nature; it does not consider any context beyond the window of $N_{(-)} + N_{(+)} + 1$ words; its implementation is straightforward, even more that of Viterbi's algorithm. The main problem is the estimation of the probabilities $p(\gamma[t]|C_{(-)}[t]\sigma[t]C_{(+)}[t])$. From a tagged corpus, these probabilities may be easily counted; in this paper, however, we propose a way of estimating them from an untagged corpus. Another problem is the large number of parameters of the model (worst case $O(|\Sigma|^{N_{(+)}+N_{(-)}+1}|\Gamma|)$); we will discuss a way to reduce the number of parameters to just $O(|\Sigma|^{N_{(+)}+N_{(-)}}|\Gamma|)$ and show that, for many applications, $N_{(-)} = N_{(+)} = 1$ is an adequate choice.

3 Training from an Untagged Corpus

The main approximation in this model is the following: we will assume that the probability of finding a certain tag $\gamma[t]$ in the center of the window depends only on the preceding context $C_{(-)}[t]$ and the succeeding context $C_{(+)}[t]$ but not on the particular word class at position t, $\sigma[t]$; that is, the probability that a word receives a certain label depends only *selectionally* on the word (the context determines the probabilities of each label, whereas the word just selects labels among those in $T(\sigma[t])$). We will denote this probability as $p_{C_{(-)}\gamma C_{(+)}}$ for short (with the position index $[t]$ dropped because of the invariance). The most probable tag $\gamma^*[t]$ is then

$$\gamma^*[t] = \operatorname*{argmax}_{\gamma \in T(\sigma[t])} p_{C_{(-)}[t]\gamma C_{(+)}[t]}, \tag{6}$$

that is, the most probable tag in that context among those corresponding to the current word class. The probabilities $p_{C_{(-)}\gamma C_{(+)}}$ are easily estimated from a tagged corpus (e.g., by counting) but estimating them from an untagged corpus involves an iterative process; instead of estimating the probability we will estimate the count $\tilde{n}_{C_{(-)}\gamma C_{(+)}}$ which can be interpreted as the effective number of times that label γ would appear in the text between contexts $C_{(-)}$ and $C_{(+)}$.

Therefore,

$$p(\gamma|C_{(-)}\sigma C_{(+)}) = \begin{cases} k_{\sigma_{(-)}\sigma\sigma_{(+)}}\tilde{n}_{C_{(-)}\gamma C_{(+)}} & \text{if } \gamma \in T(\sigma) \\ 0 & \text{otherwise} \end{cases}, \tag{7}$$

where $k_{\sigma_{(-)}\sigma\sigma_{(+)}}$ is a normalization factor

$$k_{\sigma_{(-)}\sigma\sigma_{(+)}} = \left(\sum_{\gamma' \in T(\sigma)} \tilde{n}_{C_{(-)}\gamma' C_{(+)}} \right)^{-1}. \tag{8}$$

Now, how can the counts $\tilde{n}_{C_{(-)}[t]\gamma C_{(+)}[t]}$ be estimated? If the window probabilities $p(\gamma|C_{(-)}[t]\sigma C_{(+)}[t])$ were known, they could be easily obtained from the text itself as follows:

$$\tilde{n}_{C_{(-)}\gamma C_{(+)}} = \sum_{\sigma:\gamma \in T(\sigma)} n_{C_{(-)}\sigma C_{(+)}} p(\gamma|C_{(-)}\sigma C_{(+)}), \tag{9}$$

where $n_{C_{(-)}\sigma C_{(+)}}$ is the number of times that label σ appears between contexts $C_{(-)}$ and $C_{(+)}$; that is, one would add $p(\gamma|C_{(-)}\sigma C_{(+)})$ each time a word class σ containing tag γ appears between $C_{(-)}$ and $C_{(+)}$. Equations (7) and (9) may be iteratively solved until the $\tilde{n}_{C_{(-)}\gamma C_{(+)}}$ converge. For the computation to be more efficient, one can avoid storing the probabilities $p(\gamma|C_{(-)}\sigma C_{(+)})$ by organizing the iterations around the $\tilde{n}_{C_{(-)}\gamma C_{(+)}}$ as follows, by combining eqs. (7), (8), and (9) and using an iteration index denoted with a superscript $[k]$,

$$\tilde{n}^{[k]}_{C_{(-)}\gamma C_{(+)}} = \tilde{n}^{[k-1]}_{C_{(-)}\gamma C_{(+)}} \sum_{\sigma:\gamma \in T(\sigma)} n_{C_{(-)}\sigma C_{(+)}} \left(\sum_{\gamma' \in T(\sigma)} \tilde{n}^{[k-1]}_{C_{(-)}\gamma' C_{(+)}} \right)^{-1}, \tag{10}$$

where the iteration may be easily seen as a process of successive corrections to the effective counts $\tilde{n}_{C_{(-)}\gamma C_{(+)}}$. A possible initial value is given by

$$\tilde{n}^{[0]}_{C_{(-)}\gamma C_{(+)}} = \sum_{\sigma:\gamma \in T(\sigma)} n_{C_{(-)}\sigma C_{(+)}} \frac{1}{|T(\sigma)|}, \tag{11}$$

that is, assuming that, initially, all possible tags are equally probable for each word class.

Equations (10) and (11) contain the counts $n_{C_{(-)}\sigma C_{(+)}}$ which depend on $N_{(+)} + N_{(-)} + 1$ word classes; if memory is at a premium, instead of reading the text once to count these and then iterating, the text may be read in each iteration to avoid storing the $n_{C_{(-)}\sigma C_{(+)}}$, and the $\tilde{n}^{[k]}_{C_{(-)}\gamma C_{(+)}}$ may be computed *on the fly*. Iterations proceed until a selected convergence condition has been met (e.g. a comparison of the $\tilde{n}^{[k]}_{C_{(-)}\gamma C_{(+)}}$ with respect to the $\tilde{n}^{[k-1]}_{C_{(-)}\gamma C_{(+)}}$, or the completion of a predetermined number of iterations).

4 Tagging Text: A Finite-State Tagger

Once the $\tilde{n}_{C_{(-)}\gamma C_{(+)}}$ have been computed, the winning tag for class σ in context $C_{(-)} \cdots C_{(+)}$, eq. (6), may be easily computed for all of the words in a text. Unlike with HMM [2], a sliding window PoS tagger may be turned exactly into a finite-state transducer [6]; in particular, into a Mealy machine with transitions having the form

$$\begin{array}{l}\sigma[t-N_{(-)}]\sigma[t-N_{(-)}+1]\cdots\sigma[t+N_{(+)}-1] \overset{\sigma[t+N_{(+)}]:\gamma^*}{\longrightarrow} \\ \sigma[t-N_{(-)}+1]\sigma[t-N_{(-)}+2]\cdots\sigma[t+N_{(+)}]\end{array}$$

This Mealy machine reads a text $\sigma[1]\ldots\sigma[L]$ word by word and outputs the winner tag sequence $\gamma^*[1]\ldots\gamma^*[L]$ with a delay of $N_{(+)}$ words. The resulting transducer has, in the worst case, $O(|\Sigma|^{N_{(+)}+N_{(-)}})$ states and $O(|\Sigma|^{N_{(+)}+N_{(-)}+1})$ transitions, but it may be minimized using traditional methods for finite-state transducers into a compact version of the sliding window PoS tagger, which takes into account the fact that different contexts may actually be grouped because they lead to the same disambiguation results.

5 Experiments

This section reports experiments to assess the performance of sliding-window part-of-speech using different amounts of context, compares it with that of customary Baum-Welch-trained HMM taggers [1], and describes the conversion of the resulting SWPoST taggers into finite-state machines.

The corpora we have used for training and testing is the Penn Treebank, version 3 [4, 3], which has more than one million PoS-tagged words (1014377) of English text taken from *The Wall Street Journal.* The word classes Σ of the Treebank will be taken simply to be ambiguity classes, that is, subsets of the collection of different part-of-speech tags (Γ). The Treebank has 45 different part-of-speech tags; 244261 words are ambiguous (24.08%).

The experiments use a lexicon extracted from the Penn Treebank, that is, a list of words with all the possible parts of speech observed. The exact tag given in the Treebank for each occurrence of each word is taken into account for testing but not for training. However, to simulate the effect of using a real, limited morphological analyser, we have filtered the resulting lexicon as follows:

– only the 14276 most frequent words have been kept, which ensures a 95% text coverage (i.e, 5% of the words are unknown).
– for each word, any part-of-speech tag occuring less than 5% of the time has been removed.

Using this simplified lexicon, texts in the Penn Treebank show 219 ambiguity classes. Words which are not included in our lexicon are assigned to a special ambiguity class (the *open* class) containing all tags representing parts of speech that can grow (i.e. a new word can be a noun or a verb but hardly ever a preposition).[1]

In order to train the taggers we have applied the following strategy, so that we can use as much text as possible for training: the Treebank is divided into 20 similarly-sized sections; a leaving-one-out procedure is applied, using 19 sections for training and the remaining one for testing, so that our results are the average of all 20 different train–test configurations. Figures show the average correct-tag rate only over ambiguous words (non-ambiguous words are not counted as successful disambiguations).

5.1 Effect of the Amount of Context

First of all, we show the results of a SWPoST using no context ($N_{(-)} = N_{(+)} = 0$) as a baseline, and compare them to those of a Baum-Welch-trained HMM tagger and to random tagging. As can be seen in figure 1, the performance of the SWPoST without context is not much better than random tagging. This happens because without context the SWPoST simply delivers an estimate of the most likely tag in each class. We can also observe that the HMM has an accuracy of around 61% of ambiguous words.

In order to improve the results one obviously needs to increase the context (i.e., widen the sliding window). As a first step we show the results of using a reduced context of only one word either before ($N_{(-)} = 1, N_{(+)} = 0$) or after ($N_{(-)} = 0, N_{(+)} = 1$) the current word. Figure 2 shows how that even using such a limited context the performance is more adequate. The number or trainable parameters of the SWPoST in this case is $O(|\mathit{\Sigma}||\mathit{\Gamma}|)$, slightly less than the $O(|\mathit{\Sigma}||\mathit{\Gamma}| + |\mathit{\Gamma}|^2)$ of the HMM tagger.

There is a significant difference between using as context the preceding ($N_{(-)} = 1$ and $N_{(+)} = 0$) or the succeeding ($N_{(-)} = 0$ and $N_{(+)} = 1$) word. The cognitive origin of this difference could be due to the fact that when people process language they tend to build hyphotheses based on what they have already heard or read which are used to reduce the ambiguity of words as they arrive.

[1] Our open class contains the tags `CD`, `JJ`, `JJR`, `JJS`, `NN`, `NNP`, `NNPS`, `RB`, `RBR`, `RBS`, `UH`, `VB`, `VBD`, `VBG`, `VBN`, `VBP`, and `VBZ`.

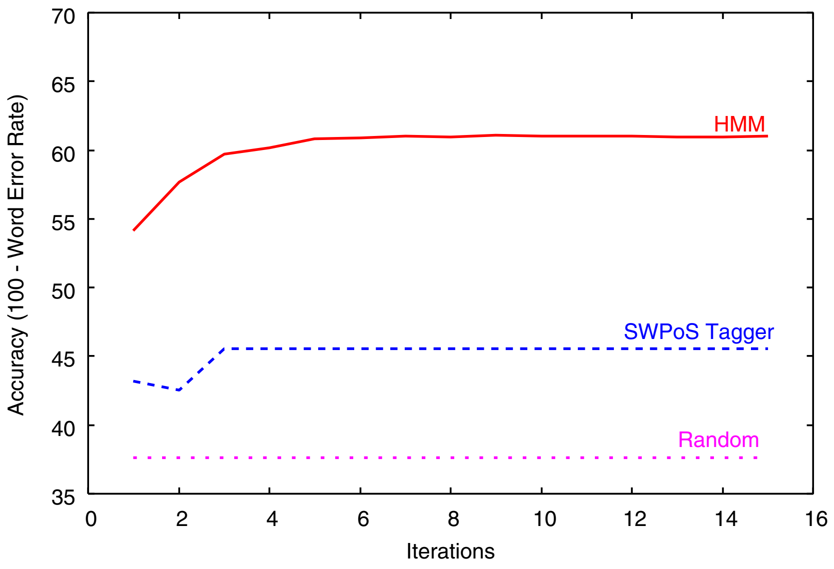

Fig. 1. Comparison between an HMM tagger, the SWPoST with no context ($N_{(-)} = N_{(+)} = 0$) and a random tagger

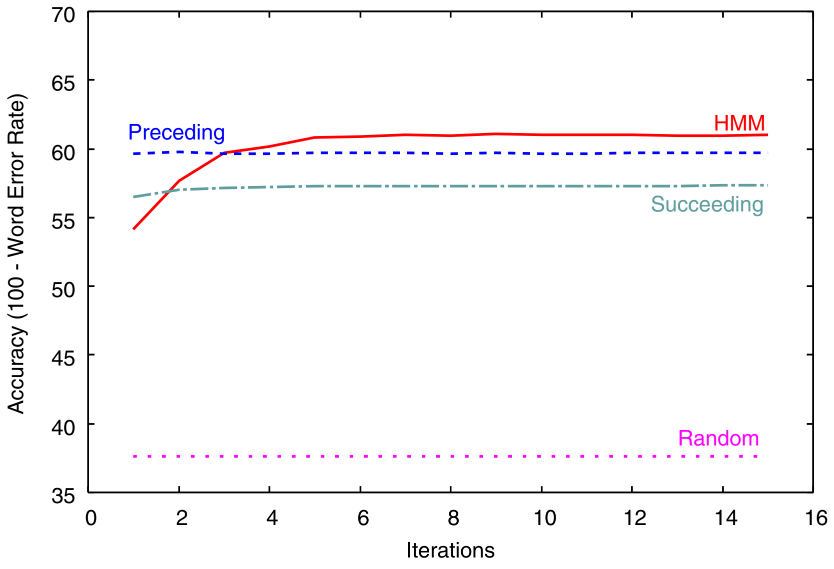

Fig. 2. Comparison between an HMM tagger and the SWPoST using $N_{(-)} = 1$ and $\mathrm{N}_{(+)}$=0 (preceding) and using $N_{(-)}$=0 and $N_{(+)}$=1 (succeeding)

The next step is increasing a bit more the size of the context until having two context words. In this case we have three different possibilities: using the two immediately preceding words ($N_{(-)} = 2$ and $N_{(+)} = 0$), using one preceeding and one succeeding word ($N_{(-)} = 1$ and $N_{(+)} = 1$), and using two succeeding words ($N_{(-)} = 0$ and $N_{(+)} = 2$). We can see the results in figure 3. The performance of the SWPoST is now much better than the HMM tagger when using a context of $N_{(-)}$=1 and $N_{(+)}$=1. However when using the two succeeding words the results are worse than with the HMM tagger, and the performance of the SWPoST with $N_{(-)}$=2 and $N_{(+)}$=0 is about as good as that of an HMM tagger.

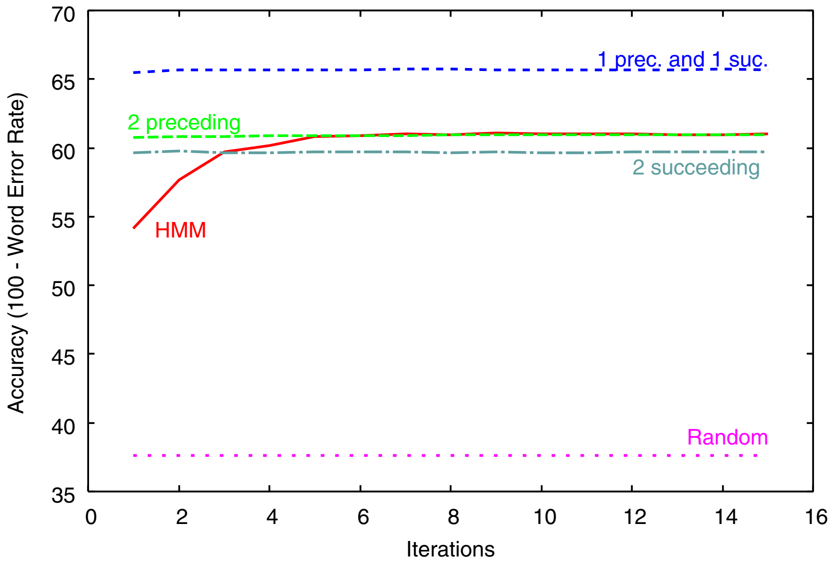

Fig. 3. Comparison between an HMM tagger and the SWPoST with using $N_{(-)}$=2 and $N_{(+)}$=0 (2 preceding) and using $N_{(-)}$=1 and $N_{(+)}$=1 (1 prec. and 1 suc.) and $N_{(-)}$=0 and $N_{(+)}$=2 (2 succeeding)

Finally, we tried increasing the context a bit more, until using three context words in all possible geometries, but the results were not as good as we expected (actually worse) due to the fact that the corpus is not large enough to allow the estimation of $O|\Gamma||\Sigma|^3)$ parameters.

The whole set of figures shows that SWPoST training usually converges after three or four iterations, which makes training very efficient in terms of time.

5.2 Finite-State Sliding-Window PoS Tagger

Once we have analysed the performance of the SWPoST we study its transformation into an equivalent finite-state transducer (FST). Given that the best results reported in the previous section correspond to using the context $N_{(-)}$=1

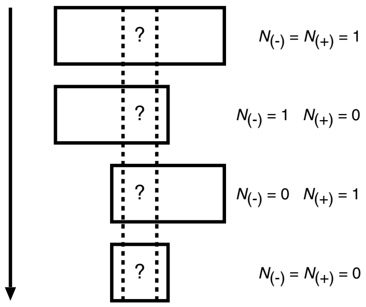

Fig. 4. Fallback strategy

and $N_{(+)}$=1, we build a FST that has a decision delay of 1 time unit, with transitions of the form

$$\sigma[t-1]\sigma[t] \xrightarrow{\sigma[t+1]:\gamma^*[t]} \sigma[t]\sigma[t+1].$$

Many of these transitions correspond to contexts that have never or hardly ever been observed in the corpus. To improve the accuracy of the tagger, a fallback strategy was applied; this strategy uses SWPoST with smaller contexts trained on the same corpus to define the output of these unseen transitions. Figure 4 shows the order of preference of the fallback strategy: if $N_{(-)} = 1, N_{(+)} = 1$ fails, the next best tagger $N_{(-)} = 1, N_{(+)} = 0$ is used; if this fails, $N_{(-)} = 0, N_{(+)} = 1$ is used, etc. The resulting FST has a slightly improved performance, reaching 67.15% accuracy for ambiguous words.

5.3 Minimization of the Finite-State SWPoST

The FST created in this way has a large number of states; customary finite-state minimization may be expected to reduce the number of states and therefore reduce memory requirements. The algorithm to build the FST generates in our case 48400 states ($|\Sigma|^2$) and 10648000 ($|\Sigma|^3$) transitions. After minimization the FST is reduced to 22137 states and 4870140 transitions. Given the large amount of ambiguity classes, minimizing to about half the size is not far from what we expected.

6 Concluding Remarks

We have shown that, as commonly-used HMM taggers, simple and intuitive sliding-window part-of-speech taggers (SWPoST) may be iteratively trained in an unsupervised manner using reasonable approximations to reduce the number of trainable parameters. The number of trainable parameters depends on the size of the sliding window. Experimental results with the Penn Treebank show that the performance of SWPoST and HMM taggers having a similar number of trainable parameters is comparable. The best results, better than those of HMM taggers, are obtained using a SWPoST using a context of one preceding and one succeeding word, for a worst-case total of 2178000 parameters (with the HMM tagger having only 11925). The SWPoST can be exactly implemented as a finite-state transducer which, after minimization, has 22137 states and 4870140 transitions. Furthermore, the functioning of SWPoST is simple and intuitive, which allows for simple implementation and maintenance; for instance, if a training error is found it is easy to manually correct a transition in the resulting FST.

We are currently studying ways to reduce further the number of states and transitions at a small price in tagging accuracy, by using probabilistic criteria to prune uncommon contexts which do not contribute significantly to the overall accuracy.

References

1. D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Third Conference on Applied Natural Language Processing. Association for Computational Linguistics. Proceedings of the Conference*, pages 133–140, Trento, Italia, 31 marzo–3 abril 1992.
2. André Kempe. Finite state transducers approximating hidden Markov models. In Philip R. Cohen and Wolfgang Wahlster, editors, *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 460–467, Somerset, New Jersey, 1997.
3. Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. The Penn Treebank: Annotating predicate argument structure. In *Proc. ARPA Human Language Technology Workshop*, pages 110–115, 1994.
4. Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: the Penn Treebank. *Computational linguistics*, 19:313–330, 1993. Reprinted in Susan Armstrong, ed. 1994, *Using large corpora*, Cambridge, MA: MIT Press, 273–290.
5. Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
6. E. Roche and Y. Schabes. Introduction. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 1–65. MIT Press, Cambridge, Mass., 1997.