PART-OF-SPEECH TAGGING USING A VARIABLE MEMORY MARKOV MODEL

Hinrich Schütze
Center for the Study of
Language and Information
Stanford, CA 94305-4115
Internet: schuetze@csli.stanford.edu

Yoram Singer
Institute of Computer Science and
Center for Neural Computation
Hebrew University, Jerusalem 91904
Internet: singer@cs.huji.ac.il

Abstract

We present a new approach to disambiguating syntactically ambiguous words in context, based on *Variable Memory Markov* (VMM) models. In contrast to fixed-length Markov models, which predict based on fixed-length histories, variable memory Markov models dynamically adapt their history length based on the training data, and hence may use fewer parameters. In a test of a VMM based tagger on the Brown corpus, 95.81% of tokens are correctly classified.

INTRODUCTION

Many words in English have several parts of speech (*POS*). For example "book" is used as a noun in "She read a book." and as a verb in "She didn't book a trip." Part-of-speech tagging is the problem of determining the syntactic part of speech of an occurrence of a word in context. In any given English text, most tokens are syntactically ambiguous since most of the high-frequency English words have several parts of speech. Therefore, a correct syntactic classification of words in context is important for most syntactic and other higher-level processing of natural language text.

Two stochastic methods have been widely used for POS tagging: fixed order Markov models and Hidden Markov models. Fixed order Markov models are used in (Church, 1989) and (Charniak et al., 1993). Since the order of the model is assumed to be fixed, a short memory (small order) is typically used, since the number of possible combinations grows exponentially. For example, assuming there are 184 different tags, as in the Brown corpus, there are $184^3 = 6,229,504$ different order 3 combinations of tags (of course not all of these will actually occur, see (Weischedel et al., 1993)). Because of the large number of parameters higher-order fixed length models are hard to estimate. (See (Brill, 1993) for a rule-based approach to incorporating higher-order information.) In a *Hidden Markov Model* (HMM) (Jelinek, 1985; Kupiec, 1992), a different state is defined for each POS tag and the transition probabilities and the output probabilities are estimated using the EM (Dempster et al., 1977) algorithm, which guarantees convergence to a local minimum (Wu, 1983). The advantage of an HMM is that it can be trained using untagged text. On the other hand, the training procedure is time consuming, and a fixed model (topology) is assumed. Another disadvantage is due to the local convergence properties of the EM algorithm. The solution obtained depends on the initial setting of the model's parameters, and different solutions are obtained for different parameter initialization schemes. This phenomenon discourages linguistic analysis based on the output of the model.

We present a new method based on *variable memory Markov* models (VMM) (Ron et al., 1993; Ron et al., 1994). The VMM is an approximation of an unlimited order Markov source. It can incorporate both the static (order 0) and dynamic (higher-order) information systematically, while keeping the ability to change the model due to future observations. This approach is easy to implement, the learning algorithm and classification of new tags are computationally efficient, and the results achieved, using simplified assumptions for the static tag probabilities, are encouraging.

VARIABLE MEMORY MARKOV MODELS

Markov models are a natural candidate for language modeling and temporal pattern recognition, mostly due to their mathematical simplicity. However, it is obvious that finite memory Markov models cannot capture the recursive nature of language, nor can they be trained effectively with long memories. The notion of *variable context length* also appears naturally in the context of universal coding (Rissanen, 1978; Rissanen and Langdon, 1981). This information theoretic notion is now known to be closely related to efficient modeling (Rissanen, 1988). The natural measure that appears in information theory is the description length, as measured by the statistical predictability via the Kullback-Leibler (KL) divergence.

The VMM learning algorithm is based on minimizing the statistical prediction error of a Markov model, measured by the instantaneous KL divergence of the following symbols, the current *statistical surprise* of the model. The memory is extended precisely when such a surprise is significant, until the overall statistical prediction of the stochastic model is sufficiently good. For the sake of simplicity, a POS tag is termed a symbol and a sequence of tags is called a string. We now briefly describe the algorithm for learning a variable memory Markov model. See (Ron et al., 1993; Ron et al., 1994) for a more detailed description of the algorithm.

We first introduce notational conventions and define some basic concepts. Let Σ be a finite alphabet. Denote by $\Sigma^\star$ the set of all strings over Σ. A string s, over $\Sigma^\star$ of length n, is denoted by $s = s_1 s_2 \ldots s_n$. We denote by $\mathbf{e}$ the empty string. The length of a string s is denoted by $|s|$ and the size of an alphabet Σ is denoted by $|\Sigma|$. Let $Prefix(s) = s_1 s_2 \ldots s_{n-1}$ denote the longest prefix of a string s, and let $Prefix^\star(s)$ denote the set of all prefixes of s, including the empty string. Similarly, $Suffix(s) = s_2 s_3 \ldots s_n$ and $Suffix^\star(s)$ is the set of all suffixes of s. A set of strings is called a suffix (prefix) free set if, $\forall s \in S : S \bigcap Suffix^\star(s) = \emptyset$ ($S \bigcap Prefix^\star(s) = \emptyset$). We call a probability measure P, over the strings in $\Sigma^\star$ *proper* if $P(\mathbf{e}) = 1$, and for every string s, $\sum_{\sigma \in \Sigma} P(s\sigma) = P(s)$. Hence, for every prefix free set S, $\sum_{s \in S} P(s) \leq 1$, and specifically for every integer $n \geq 0$, $\sum_{s \in \Sigma^n} P(s) = 1$.

A prediction suffix tree T over Σ, is a tree of degree $|\Sigma|$. The edges of the tree are labeled by symbols from Σ, such that from every internal node there is at most one outgoing edge labeled by each symbol. The nodes of the tree are labeled by pairs (s, γ_s) where s is the string associated with the walk starting from that node and ending in the root of the tree, and $\gamma_s : \Sigma \rightarrow [0,1]$ is the *output probability function* of s satisfying $\sum_{\sigma \in \Sigma} \gamma_s(\sigma) = 1$. A prediction suffix tree induces probabilities on arbitrarily long strings in the following manner. The probability that T generates a string $w = w_1 w_2 \ldots w_n$ in Σ^n, denoted by $P_T(w)$, is $\Pi_{i=1}^{n} \gamma_{s^{i-1}}(w_i)$, where $s^0 = \mathbf{e}$, and for $1 \leq i \leq n-1$, s^j is the string labeling the *deepest* node reached by taking the walk corresponding to $w_1 \ldots w_i$ starting at the root of T. By definition, a prediction suffix tree induces a proper measure over $\Sigma^\star$, and hence for every prefix free set of strings $\{w^1, \ldots, w^m\}$, $\sum_{i=1}^{m} P_T(w^i) \leq 1$, and specifically for $n \geq 1$, then $\sum_{s \in \Sigma^n} P_T(s) = 1$.

A *Probabilistic Finite Automaton (PFA) A* is a 5-tuple $(Q, \Sigma, \tau, \gamma, \pi)$, where Q is a finite set of *n states*, Σ is an *alphabet* of size k, $\tau : Q \times \Sigma \rightarrow Q$ is the *transition function*, $\gamma : Q \times \Sigma \rightarrow [0,1]$ is the *output probability function*, and $\pi : Q \rightarrow [0,1]$ is the probability distribution over the *start states*. The functions γ and π must satisfy the following requirements: for every $q \in Q$, $\sum_{\sigma \in \Sigma} \gamma(q, \sigma) = 1$, and $\sum_{q \in Q} \pi(q) = 1$. The probability that A generates a string $s = s_1 s_2 \ldots s_n \in \Sigma^n$ is $P_A(s) = \sum_{q^0 \in Q} \pi(q^0) \prod_{i=1}^{n} \gamma(q^{i-1}, s_i)$, where $q^{i+1} = \tau(q^i, s_i)$. τ can be extended to be defined on $Q \times \Sigma^\star$ as follows: $\tau(q, s_1 s_2 \ldots s_l) = \tau(\tau(q, s_1 \ldots s_{l-1}), s_l) = \tau(\tau(q, Prefix(s)), s_l)$. The distribution over the states, π, can be replaced by a single start state, denoted by ϵ such that $\tau(\epsilon, s) = \pi(q)$, where s is the label of the state q. Therefore, $\pi(\epsilon) = 1$ and $\pi(q) = 0$ if $q \neq \epsilon$.

For POS tagging, we are interested in learning a sub-class of finite state machines which have the following property. Each state in a machine M belonging to this sub-class is labeled by a string of length at most L over Σ, for some $L \geq 0$. The set of strings labeling the states is suffix free. We require that for every two states $q^1, q^2 \in Q$ and for every symbol $\sigma \in \Sigma$, if $\tau(q^1, \sigma) = q^2$ and q^1 is labeled by a string s^1, then q^2 is labeled by a string s^2 which is a suffix of $s^1 \cdot \sigma$. Since the set of strings labeling the states is suffix free, if there exists a string having this property then it is unique. Thus, in order that τ be well defined on a given set of string S, not only must the set be suffix free, but it must also have the property, that for every string s in the set and every symbol σ, there exists a string which is a suffix of $s\sigma$. For our convenience, from this point on, if q is a state in Q then q will also denote the string labeling that state.

A special case of these automata is the case in which Q includes *all* $|\Sigma|^L$ strings of length L. These automata are known as *Markov processes of order L*. We are interested in learning automata for which the number of states, n, is much smaller than $|\Sigma|^L$, which means that few states have long memory and most states have a short one. We refer to these automata as *variable memory Markov* (VMM) processes. In the case of Markov processes of order L, the identity of the states (i.e. the identity of the strings labeling the states) is known and learning such a process reduces to approximating the output probability function.

Given a sample consisting of m POS tag sequences of lengths $l_1, l_2, \ldots, l_m$ we would like to find a prediction suffix tree that will have the same statistical properties as the sample and thus can be used to predict the next outcome for sequences generated by the same source. At each stage we can transform the tree into a variable memory Markov process. The key idea is to iteratively build a prediction tree whose probability measure equals the empirical probability measure calculated from the sample.

We start with a tree consisting of a single node and add nodes which we have reason to believe should be in the tree. A node σs, must be added to the tree if it statistically differs from its parent node s. A natural measure to check the statistical difference is the *relative entropy* (also known as the Kullback-Leibler (KL) divergence) (Kullback, 1959), between the conditional probabilities $P(\cdot|s)$ and $P(\cdot|\sigma s)$. Let X be an observation space and P_1, P_2 be probability measures over X then the KL divergence between P_1 and P_2 is, $D_{KL}(P_1||P_2) = \sum_{x \in X} P_1(x) \log \frac{P_1(x)}{P_2(x)}$. In our case, the KL divergence measures how much additional information is gained by using the suffix σs for prediction instead of the shorter suffix s. There are cases where the statistical difference is large yet the probability of observing the suffix σs itself is so small that we can neglect those cases. Hence we weigh the statistical error by the prior probability of observing σs. The statistical error measure in our case is,

$$\begin{aligned} & Err(\sigma s, s) \\ & \quad = P(\sigma s) D_{KL}(P(\cdot|\sigma s)||P(\cdot|s)) \\ & \quad = P(\sigma s) \textstyle\sum_{\sigma' \in \Sigma} P(\sigma'|\sigma s) \log \frac{P(\sigma'|\sigma s)}{P(\sigma'|s)} \\ & \quad = \textstyle\sum_{\sigma' \in \Sigma} P(\sigma s \sigma') \log \frac{P(\sigma s \sigma')}{P(\sigma'|s) P(\sigma s)} \quad . \end{aligned}$$

Therefore, a node σs is added to the tree if the statistical difference (defined by $Err(\sigma s, s)$) between the node and its parrent s is larger than a predetermined accuracy ϵ. The tree is grown level by level, adding a son of a given leaf in the tree whenever the statistical error is large. The problem is that the requirement that a node statistically differs from its parent node is a necessary condition for belonging to the tree, but is not sufficient. The leaves of a prediction suffix tree must differ from their parents (or they are redundant) but internal nodes might not have this property. Therefore, we must continue testing further potential descendants of the leaves in the tree up to depth L. In order to avoid exponential grow in the number of strings tested, we do not test strings which belong to branches which are reached with small probability. The set of strings, tested at each step, is denoted by S, and can be viewed as a kind of frontier of the growing tree T.

USING A VMM FOR POS TAGGING

We used a tagged corpus to train a VMM. The syntactic information, i.e. the probability of a specific word belonging to a tag class, was estimated using *maximum likelihood* estimation from the individual word counts. The states and the transition probabilities of the Markov model were determined by the learning algorithm and tag output probabilities were estimated from word counts (the static information present in the training corpus). The whole structure, for two states, is depicted in Fig. 1. S_i and S_{i+1} are strings of tags corresponding to states of the automaton. $P(t_i|S_i)$ is the probability that tag t_i will be output by state S_i and $P(t_{i+1}|S_{i+1})$ is the probability that the next tag t_{i+1} is the output of state S_{i+1}.

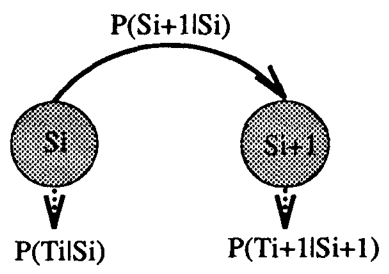

Figure 1: The structure of the VMM based POS tagger.

When tagging a sequence of words $w_{1,n}$, we want to find the tag sequence $t_{1,n}$ that is most likely for $w_{1,n}$. We can maximize the joint probability of $w_{1,n}$ and $t_{1,n}$ to find this sequence:[1]

$$\begin{aligned} \mathcal{T}(w_{1,n}) &= \arg\max_{t_{1,n}} P(t_{1,n}|w_{1,n}) \\ &= \arg\max_{t_{1,n}} \frac{P(t_{1,n}, w_{1,n})}{P(w_{1,n})} \\ &= \arg\max_{t_{1,n}} P(t_{1,n}, w_{1,n}) \end{aligned}$$

$P(t_{1,n}, w_{1,n})$ can be expressed as a product of conditional probabilities as follows:

$$\begin{aligned} & P(t_{1,n}, w_{1,n}) = \\ & \quad P(t_1)P(w_1|t_1)P(t_2|t_1, w_1)P(w_2|t_{1,2}, w_1) \\ & \quad \ldots P(t_n|t_{1,n-1}, w_{1,n-1})P(w_n|t_{1,n}, w_{1,n-1}) \\ & \quad = \prod_{i=1}^{n} P(t_i|t_{1,i-1}, w_{1,i-1})P(w_i|t_{1,i}, w_{1,i-1}) \end{aligned}$$

With the simplifying assumption that the probability of a tag only depends on previous tags and that the probability of a word only depends on its tags, we get:

$$P(t_{1,n}, w_{1,n}) = \prod_{i=1}^{n} P(t_i|t_{1,i-1})P(w_i|t_i)$$

Given a variable memory Markov model $\mathbf{M}$, $P(t_i|t_{1,i-1})$ is estimated by $P(t_i|S_{i-1}, \mathbf{M})$ where

[1]Part of the following derivation is adapted from (Charniak et al., 1993).

$S_i = \tau(\epsilon, t_{1,i})$, since the dynamics of the sequence are represented by the transition probabilities of the corresponding automaton. The tags $t_{1,n}$ for a sequence of words $w_{1,n}$ are therefore chosen according to the following equation using the Viterbi algorithm:

$$\mathcal{T}_{\mathrm{M}}(w_{1,n}) = \arg\max_{t_{1,n}} \prod_{i=1}^{n} P(t_i|S_{i-1}, \mathrm{M})P(w_i|t_i)$$

We estimate $P(w_i|t_i)$ indirectly from $P(t_i|w_i)$ using Bayes' Theorem:

$$P(w_i|t_i) = \frac{P(w_i)P(t_i|w_i)}{P(t_i)}$$

The terms $P(w_i)$ are constant for a given sequence w_i and can therefore be omitted from the maximization. We perform a maximum likelihood estimation for $P(t_i)$ by calculating the relative frequency of t_i in the training corpus. The estimation of the static parameters $P(t_i|w_i)$ is described in the next section.

We trained the variable memory Markov model on the Brown corpus (Francis and Kučera, 1982), with every tenth sentence removed (a total of 1,022,462 tags). The four stylistic tag modifiers "FW" (foreign word), "TL" (title), "NC" (cited word), and "HL" (headline) were ignored reducing the complete set of 471 tags to 184 different tags.

The resulting automaton has 49 states: the null state (ϵ), 43 first order states (one symbol long) and 5 second order states (two symbols long). This means that 184-43=141 states were not (statistically) different enough to be included as separate states in the automaton. An analysis reveals two possible reasons. Frequent symbols such as "ABN" ("half", "all", "many" used as prequantifiers, e.g. in "many a younger man") and "DTI" (determiners that can be singular or plural, "any" and "some") were not included because they occur in a variety of diverse contexts or often precede unambiguous words. For example, when tagged as "ABN" "half", "all", and "many" tend to occur before the unambiguous determiners "a", "an" and "the".

Some rare tags were not included because they did not improve the optimization criterion, minimum description length (measured by the KL-divergence). For example, "HVZ*" ("hasn't") is not a state although a following "- ed" form is always disambiguated as belonging to class "VBN" (past participle). But since this is a rare event, describing all "HVZ* VBN" sequences separately is cheaper than the added complexity of an automaton with state "HVZ*". We in fact lost some accuracy in tagging because of the optimization criterion: Several "-ed" forms after forms of "have" were mistagged as "VBD" (past tense).

transition to	one-symbol state	two-symbol state
NN	JJ: 0.45	AT JJ: 0.69
IN	JJ: 0.06	AT JJ: 0.004
IN	NN: 0.27	AT NN: 0.35
.	NN: 0.14	AT NN: 0.10
NN	VBN: 0.08	AT VBN: 0.48
IN	VBN: 0.35	AT VBN: 0.003
NN	CC: 0.12	JJ CC: 0.04
JJ	CC: 0.09	JJ CC: 0.58
VB	RB: 0.05	MD RB: 0.48
VBN	RB: 0.08	MD RB: 0.0009

Table 1: States for which the statistical prediction is significantly different when using a longer suffix for prediction. Those states are identified automatically by the VMM learning algorithm. A better prediction and classification of POS-tags is achieved by adding those states with only a small increase in the computation time.

The two-symbol states were "AT JJ", "AT NN", "AT VBN", "JJ CC", and "MD RB" (article adjective, article noun, article past participle, adjective conjunction, modal adverb). Table 1 lists two of the largest differences in transition probabilities for each state. The varying transition probabilities are based on differences between the syntactic constructions in which the two competing states occur. For example, adjectives after articles ("AT JJ") are almost always used attributively which makes a following preposition impossible and a following noun highly probable, whereas a predicative use favors modifying prepositional phrases. Similarly, an adverb preceded by a modal ("MD RB") is followed by an infinitive ("VB") half the time, whereas other adverbs occur less often in pre-infinitival position. On the other hand, a past participle is virtually impossible after "MD RB" whereas adverbs that are not preceded by modals modify past participles quite often.

While it is known that Markov models of order 2 give a slight improvement over order-1 models (Charniak et al., 1993), the number of parameters in our model is much smaller than in a full order-2 Markov model (49*184 = 9016 vs. 184*184*184 = 6,229,504).

ESTIMATION OF THE STATIC PARAMETERS

We have to estimate the conditional probabilities $P(t^i|w^j)$, the probability that a given word w^j will appear with tag t^i, in order to compute the static parameters $P(w^j|t^i)$ used in the tagging equations described above. A first approximation would be to use the maximum likelihood estimator:

$$P(t^i|w^j) = \frac{C(t^i, w^j)}{C(w^j)}$$

where $C(t^i, w^j)$ is the number of times t^i is tagged as w^j in the training text and $C(w^j)$ is the number of times w^j occurs in the training text. However, some form of smoothing is necessary, since any new text will contain new words, for which $C(w^j)$ is zero. Also, words that are rare will only occur with some of their possible parts of speech in the training text. One solution to this problem is Good-Turing estimation:

$$P(t^i|w^j) = \frac{C(t^i, w^j) + 1}{C(w^j) + I}$$

where I is the number of tags, 184 in our case. It turns out that Good-Turing is not appropriate for our problem. The reason is the distinction between closed-class and open-class words. Some syntactic classes like verbs and nouns are productive, others like articles are not. As a consequence, the probability that a new word is an article is zero, whereas it is high for verbs and nouns. We need a smoothing scheme that takes this fact into account.

Extending an idea in (Charniak et al., 1993), we estimate the probability of *tag conversion* to find an adequate smoothing scheme. Open and closed classes differ in that words often add a tag from an open class, but rarely from a closed class. For example, a word that is first used as a noun will often be used as a verb subsequently, but closed classes such as possessive pronouns ("my", "her", "his") are rarely used with new syntactic categories after the first few thousand words of the Brown corpus. We only have to take stock of these "tag conversions" to make informed predictions on new tags when confronted with unseen text. Formally, let $W_l^{i,\neg k}$ be the set of words that have been seen with t^i, but not with t^k in the training text up to word w_l. Then we can estimate the probability that a word with tag t^i will later be seen with tag t^k as the proportion of words allowing tag t^i but not t^k that later add t^k:

$$P_{lm}(i \rightarrow k) = \frac{|\{n | l < n \leq m \wedge w_n \in W_l^{i,\neg k} \cap W_{n-1}^{i,\neg k} \wedge t_n = t^k\}|}{|W_l^{i,\neg k}|}$$

This formula also applies to words we haven't seen so far, if we regard such words as having occurred with a special tag "U" for "unseen". (In this case, $W_l^{U,\neg k}$ is the set of words that haven't occurred up to l.) $P_{lm}(U \rightarrow k)$ then estimates the probability that an unseen word has tag t^k. Table 2 shows the estimates of tag conversion we derived from our training text for $l = 1022462 - 100000, m = 1022462$, where 1022462 is the number of words in the training text. To avoid sparse data problems we assumed zero probability for types of tag conversion with less than 100 instances in the training set.

tag conversion	estimated probability
U → NN	0.29
U → JJ	0.13
U → NNS	0.12
U → NP	0.08
U → VBD	0.07
U → VBG	0.07
U → VBN	0.06
U → VB	0.05
U → RB	0.05
U → VBZ	0.01
U → NP$	0.01
VBD → VBN	0.09
VBN → VBD	0.05
VB → NN	0.05
NN → VB	0.01

Table 2: Estimates for tag conversion

Our smoothing scheme is then the following heuristic modification of Good-Turing:

$$P(t^i|w^j) = \frac{C(t^i, w^j) + \sum_{k_1 \in T_j} P_{lm}(k_1 \rightarrow i)}{C(w^j) + \sum_{k_1 \in T_j, k_2 \in T} P_{lm}(k_1 \rightarrow k_2)}$$

where T_j is the set of tags that w^j has in the training set and T is the set of all tags. This scheme has the following desirable properties:

- As with Good-Turing, smoothing has a small effect on estimates that are based on large counts.
- The difference between closed-class and open-class words is respected: The probability for conversion to a closed class is zero and is not affected by smoothing.
- Prior knowledge about the probabilities of conversion to different tag classes is incorporated. For example, an unseen word w^j is five times as likely to be a noun than an adverb. Our estimate for $P(t^i|w^j)$ is correspondingly five times higher for "NN" than for "RB".

ANALYSIS OF RESULTS

Our result on the test set of 114392 words (the tenth of the Brown corpus not used for training) was 95.81%. Table 3 shows the 20 most frequent errors.

Three typical examples for the most common error (tagging nouns as adjectives) are "Communist", "public" and "homerun" in the following sentences.

VMM: correct:	JJ	VBN	NN	VBD	IN	CS	NP	RP	QL	RB	VB	VBG
NN	259						102	100			69	66
VBD		228										
NNS			227									
VBN				219								
JJ			165							71		
VB			142									
CS					112							
NP	110		194									
IN						103						
VBG			94									
RB	63				63				76			
QL										64		

Table 3: Most common errors.

- the Cuban fiasco and the Communist military victories in Laos
- to increase public awareness of the movement
- the best homerun hitter

The words "public" and "communist" can be used as adjectives or nouns. Since in the above sentences an adjective is syntactically more likely, this was the tagging chosen by the VMM. The noun "homerun" didn't occur in the training set, therefore the priors for unknown words biased the tagging towards adjectives, again because the position is more typical of an adjective than of a noun.

Two examples of the second most common error (tagging past tense forms ("VBD") as past participles ("VBN")) are "called" and "elected" in the following sentences:

- the party called for government operation of all utilities
- When I come back here after the November election you'll think, you're my man – elected.

Most of the VBD/VBN errors were caused by words that have a higher prior for "VBN" so that in a situation in which both forms are possible according to local syntactic context, "VBN" is chosen. More global syntactic context is necessary to find the right tag "VBD" in the first sentence. The second sentence is an example for one of the tagging mistakes in the Brown corpus, "elected" is clearly used as a past participle, not as a past tense form.

Comparison with other Results

Charniak et al.'s result of 95.97% (Charniak et al., 1993) is slightly better than ours. This difference is probably due to the omission of rare tags that permit reliable prediction of the following tag (the case of "HVZ*" for "hasn't").

Kupiec achieves up to 96.36% correctness (Kupiec, 1992), without using a tagged corpus for training as we do. But the results are not easily comparable with ours since a lexicon is used that lists only possible tags. This can result in increasing the error rate when tags are listed in the lexicon that do not occur in the corpus. But it can also decrease the error rate when errors due to bad tags for rare words are avoided by looking them up in the lexicon. Our error rate on words that do not occur in the training text is 57%, since only the general priors are used for these words in decoding. This error rate could probably be reduced substantially by incorporating outside lexical information.

DISCUSSION

While the learning algorithm of a VMM is efficient and the resulting tagging algorithm is very simple, the accuracy achieved is rather moderate. This is due to several reasons. As mentioned in the introductory sections, any finite memory Markov model cannot capture the recursive nature of natural language. The VMM can accommodate longer statistical dependencies than a traditional full-order Markov model, but due to its Markovian nature long-distance statistical correlations are neglected. Therefore, a VMM based tagger can be used for pruning many of the tagging alternatives using its prediction probability, but not as a complete tagging system. Furthermore, the VMM power can be better utilized in low level language processing tasks such as cleaning up corrupted text as demonstrated in (Ron et al., 1993).

We currently investigate other stochastic models that can accommodate long distance statistical correlation (see (Singer and Tishby, 1994) for preliminary results). However, there are theoretical clues that those models are much harder to learn (Kearns et al., 1993), including HMM based models (Abe and Warmuth, 1992).

Another drawback of the current tagging scheme is the independence assumption of the underlying tags and the observed words, and the adhoc estimation of the static probabilities. We are pursuing a systematic scheme to estimate those probabilities based on Bayesian statistics, by assigning a discrete probability distribution, such as the Dirichlet distribution (Berger, 1985), to each tag class. The a-posteriori probability estimation of the individual words can be estimated from the word counts and the tag class priors. Those priors can be modeled as a mixture of Dirichlet distributions (Antoniak, 1974), where each mixture component would correspond to a different tag class. Currently we estimate the state transition probabilities from the conditional counts assuming a uniform prior. The same technique can be used to estimate those parameters as well.

ACKNOWLEDGMENT

Part of this work was done while the second author was visiting the Department of Computer and Information Sciences, University of California, Santa-Cruz, supported by NSF grant IRI-9123692.

We would like to thank Jan Pedersen and Naftali Tishby for helpful suggestions and discussions of this material. Yoram Singer would like to thank the Charles Clore foundation for supporting this research. We express our appreciation to faculty and students for the stimulating atmosphere at the 1993 Connectionist Models Summer School at which the idea for this paper took shape.

References

N. Abe and M. Warmuth, *On the computational complexity of approximating distributionsby probabilistic automata*, Machine Learning, Vol. 9, pp. 205–260, 1992.

C. Antoniak, *Mixture of Dirichlet processes with applications to Bayesian nonparametric problems*, Annals of Statistics, Vol. 2, pp. 1152–174, 1974.

J. Berger, *Statistical decision theory and Bayesian analysis*, New-York: Springer-Verlag, 1985.

E. Brill. Automatic grammar induction and parsing free text: A transformation-based approach. In *Proceedings of ACL 31*, pp. 259–265, 1993.

E. Charniak, Curtis Hendrickson, Neil Jacobson, and Mike Perkowitz, *Equations for Part-of-Speech Tagging*, Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 784–789, 1993.

K.W. Church, *A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text*, Proceedings of ICASSP, 1989.

A. Dempster, N. Laird, and D. Rubin, *Maximum Likelihood estimation from Incomplete Data via the EM algorithm*, J. Roy. Statist. Soc., Vol. 39(B), pp. 1–38, 1977.

W.N. Francis and F. Kučera, *Frequency Analysis of English Usage*, Houghton Mifflin, Boston MA, 1982.

F. Jelinek, *Robust part-of-speech tagging using a hidden Markov model*, IBM Tech. Report, 1985.

M. Kearns, Y. Mansour, D. Ron, R. Rubinfeld, R. Schapire, L. Sellie, *On the Learnability of Discrete Distributions*, The 25th Annual ACM Symposium on Theory of Computing, 1994.

S. Kullback, *Information Theory and Statistics*, New-York: Wiley, 1959.

J. Kupiec, *Robust part-of-speech tagging using a hidden Markov model*, Computer Speech and Language, Vol. 6, pp. 225–242, 1992.

L.R. Rabiner and B. H. Juang, *An Introduction to Hidden Markov Models*, IEEE ASSP Magazine, Vol. 3, No. 1, pp. 4–16, 1986.

J. Rissanen, *Modeling by shortest data discription*, Automatica, Vol. 14, pp. 465–471, 1978.

J. Rissanen, *Stochastic complexity and modeling*, The Annals of Statistics, Vol. 14, No. 3, pp. 1080–1100, 1986.

J. Rissanen and G. G. Langdon, *Universal modeling and coding*, IEEE Trans. on Info. Theory, IT-27, No. 3, pp. 12–23, 1981.

D. Ron, Y. Singer, and N. Tishby, *The power of Amnesia*, Advances in Neural Information Processing Systems 6, 1993.

D. Ron, Y. Singer, and N. Tishby, *Learning Probabilistic Automata with Variable Memory Length*, Proceedings of the 1994 Workshop on Computational Learning Theory, 1994.

Y. Singer and N. Tishby, *Inferring Probabilistic Acyclic Automata Using the Minimum Description Length Principle*, Proceedings of IEEE Intl. Symp. on Info. Theory, 1994.

R. Weischedel, M. Meteer, R. Schwartz, L. Ramshaw, and J. Palmucci. Coping with ambiguity and unknown words through probabilistic models. *Computational Linguistics*, 19(2):359–382, 1993.

J. Wu, *On the convergence properties of the EM algorithm*, Annals of Statistics, Vol. 11, pp. 95–103, 1983.