An SVM Based Voting Algorithm with Application to Parse Reranking

Libin Shen and **Aravind K. Joshi**
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA 19104, USA
{libin,joshi}@linc.cis.upenn.edu

Abstract

This paper introduces a novel Support Vector Machines (SVMs) based voting algorithm for reranking, which provides a way to solve the sequential models indirectly. We have presented a risk formulation under the PAC framework for this voting algorithm. We have applied this algorithm to the parse reranking problem, and achieved labeled recall and precision of $89.4\%/89.8\%$ on WSJ section 23 of Penn Treebank.

1 Introduction

Support Vector Machines (SVMs) have been successfully used in many machine learning tasks. Unlike the error-driven algorithms, SVMs search for the hyperplane that separates a set of training samples that contain two distinct classes and maximizes the margin between these two classes. The ability to maximize the margin is believed to be the reason for SVMs' superiority over other classifiers. In addition, SVMs can achieve high performance even with input data of high dimensional feature space, especially because of the use of the "kernel trick".

However, the incorporation of SVMs into sequential models remains a problem. An obvious reason is that the output of an SVM is the distance to the separating hyperplane, but not a probability. A possible solution to this problem is to map SVMs' results into probabilities through a Sigmoid function, and use Viterbi search to combine those probabilities (Platt, 1999). However, this approach conflicts with SVMs' purpose of achieving the so-called *global* optimization[1]. First, this approach may constrain SVMs to local features because of the left-to-right scanning strategy. Furthermore, like other non-generative Markov models, it suffers from the so-called *label bias problem*, which means that the transitions leaving a given state compete only against each other, rather than against all other transitions in the model (Lafferty et al., 2001). Intuitively, it is the local normalization that results in the label bias problem.

One way of using discriminative machine learning algorithms in sequential models is to rerank the n-best outputs of a generative system. Reranking uses global features as well as local features, and does not make local normalization. If the output set is large enough, the reranking approach may help to alleviate the impact of the label bias problem, because the victim parses (i.e. those parses which get penalized due to the label bias problem) will have a chance to take part in the reranking.

In recent years, reranking techniques have been successfully applied to the so-called history-based models (Black et al., 1993), especially to parsing (Collins, 2000; Collins and Duffy, 2002). In a history-based model, the current decision depends on the decisions made previously. Therefore, we may regard parsing as a special form of *sequential model* without losing generality.

Collins (2000) has proposed two reranking algorithms to rerank the output of an existing parser (Collins, 1999, Model 2). One is based on Markov Random Fields, and the other is based on a boosting approach. In (Collins and Duffy, 2002), the use of Voted Perceptron (VP) (Freund and Schapire, 1999) for the parse reranking problem has been described. In that paper, the tree kernel (Collins and Duffy, 2001) has been used to efficiently count the number of common subtrees as described in (Bod, 1998).

In this paper we will follow the reranking approach. We describe a novel SVM-based voting algorithm for reranking. It provides an alternative way of using a large margin classifier for sequential models. Instead of using the parse tree itself as a training sample, we use a pair of parse trees as a sample, which is analogous to the preference relation used in the context of ordinal regression

[1] By *global* we mean the use of quadratic optimization in margin maximization.

(Herbrich et al., 2000). Furthermore, we justify the algorithm through a modification of the proof of the large margin rank boundaries for ordinal regression. We then apply this algorithm to the parse reranking problem.

1.1 A Short Introduction of SVMs

In this section, we give a short introduction of Support Vector Machines. We follow (Vapnik, 1998)'s definition of SVMs. For each training sample $(y_i, \mathbf{x_i})$, y_i represents its class, and $\mathbf{x_i}$ represents its input vector defined on a d-dimensional space. Suppose the training samples $\{(y_1, \mathbf{x_1}), ..., (y_n, \mathbf{x_n})\}$ ($\mathbf{x_i} \in \mathcal{R}^d$, $y_i \in \{-1, 1\}$) can be separated by a hyperplane H: $(\mathbf{x} \bullet \mathbf{w}) + b = 0$, which means

$$y_i((\mathbf{x_i} \bullet \mathbf{w}) + b) \geq 1, \tag{1}$$

where $\mathbf{w}$ is normal to the hyperplane. To train an SVM is equivalent to searching for the *optimal separating hyperplane* that separates the training data without error and maximizes the *margin* between two classes of samples. It can be shown that maximizing the margin is equivalent to minimizing $||\mathbf{w}||^2$.

In order to handle linearly non-separable cases, we introduce a positive slack variable ξ_i for each sample $(y_i, \mathbf{x_i})$. Then training can be reduced to the following Quadratic Programming (QP) problem.
Maximize:

$$L_D(\alpha) \equiv \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j (\mathbf{x_i} \bullet \mathbf{x_j}) \tag{2}$$

subject to: $0 \leq \alpha_i \leq C$ and $\sum_i \alpha_i y_i = 0$,
where $\alpha_i (i = 1...l)$ are the Lagrange multipliers, l is the total number of training samples, and C is a weighting parameter for mis-classification.

Since linearly non-separable samples may become separable in a high-dimensional space, SVMs employ the "kernel trick" to implicitly separate training samples in a high-dimensional feature space. Let $\Phi : \mathcal{R}^d \mapsto \mathcal{R}^h$ be a function that maps a d-dimensional input vector $\mathbf{x}$ to an h-dimensional feature vector $\Phi(\mathbf{x})$. In order to search for the optimal separating hyperplane in the higher-dimensional feature space, we only need to substitute $\Phi(\mathbf{x_i}) \bullet \Phi(\mathbf{x_j})$ with $\mathbf{x_i} \bullet \mathbf{x_j}$ in formula (2).

If there is a function K, such that $K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i}) \bullet \Phi(\mathbf{x_j})$, we don't need to compute $\Phi(\mathbf{x_i})$ explicitly. K is called a *kernel* function. Thus during the training phase we need to solve the following QP problem.
Maximize:

$$L_D(\alpha) \equiv \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{l} y_i y_j \alpha_i \alpha_j K(\mathbf{x_i}, \mathbf{x_j}). \tag{3}$$

subject to: $0 \leq \alpha_i \leq C$, and $\sum_i \alpha_i y_i = 0$.

Let $\mathbf{x}$ be a test vector, the decision function is

$$f(\mathbf{x}) = sgn(\sum_{j=1}^{N_s} \alpha_j y_j K(\mathbf{s_j}, \mathbf{x}) + b) \tag{4}$$

where $\mathbf{s_j}$ is a training vector whose corresponding Lagrange multiplier $\alpha_j > 0$. $\mathbf{s_j}$ is called a support vector. N_s is the total number of the support vectors. According to (4), the decision function only depends on the support vectors.

It is worth noting that not any function K can be used as a kernel. We call function $K : \mathcal{R}^d \times \mathcal{R}^d \mapsto \mathcal{R}$ a well-defined kernel if and only if there is a mapping function $\Phi : \mathcal{R}^d \mapsto \mathcal{R}^h$ such that, for any $\mathbf{x_i}, \mathbf{x_j} \in \mathcal{R}^d$, $K(\mathbf{x_i}, \mathbf{x_j}) = \Phi(\mathbf{x_i}) \bullet \Phi(\mathbf{x_j})$. One way of testing whether a function is a well-defined kernel is to use the Mercer's theorem (Vapnik, 1998) by utilizing the positive semidefinteness property. However, as far as a discrete kernel is concerned, there is a more convenient way to show that a function is a well-defined kernel. This is achieved by showing that a function K is a kernel by finding the corresponding mapping function Φ. This method was used in the proof of the string subsequence kernel (Cristianini and Shawe-Tayor, 2000) and the tree kernel (Collins and Duffy, 2001).

1.2 Large Margin Classifiers

SVMs are called large margin classifiers because they search for the hyperplane that maximizes the margin. The validity of the large margin method is guaranteed by the theorems of Structural Risk Minimization (SRM) under Probably Approximately Correct (PAC) framework[2]; test error is related to training data error, number of training samples and the capacity of the learning machine (Smola et al., 2000).

Vapnik-Chervonenkis (VC) dimension (Vapnik, 1999), as well as some other measures, is used to estimate the complexity of the hypothesis space, or the capacity of the learning machine. The drawback of VC dimension is that it ignores the structure of the mapping from training samples to hypotheses, and concentrates solely on the range of the possible outputs of the learning machine (Smola et al., 2000). In this paper we will use another measure, the so-called Fat Shattering Dimension (Shawe-Taylor et al., 1998), which is shown to be more accurate than VC dimension (Smola et al., 2000), to justify our voting algorithm,

Let F be a family of hypothesis functions. The fat shattering dimension of F is a function from margin ρ to the maximum number of samples such that any subset of

[2]SVM's theoretical accuracy is much lower than their actual performance. The ability to maximize the margin is believed to be the reason for SVMs' superiority over other classifiers.

these samples can be classified with margin ρ by a function in F. An upper bound of the expected error is given in Theorem 1 below (Shawe-Taylor et al., 1998). We will use this theorem to justify the new voting algorithm.

Theorem 1 *Consider a real-valued function class F having fat-shattering function bounded above by the function $afat : \mathcal{R} \rightarrow \mathcal{N}$ which is continuous from the right. Fix $\theta \in R$. If a learner correctly classifies m independently generated examples z with $h = T_\theta(f) \in T_\theta(F)$ such that $er_z(h) = 0$ and $\rho = \min|f(x_i) - \theta|$, then with confidence $1 - \delta$ the expected error of h is bounded from above by*

$$\frac{2}{m}(k\log(\frac{8em}{k})\log(32m) + \log(\frac{8m}{\delta})) \quad (5)$$

where $k = afat(\rho/8)$.

2 A New SVM-based Voting Algorithm

Let x_{ij} be the jth candidate parse for the ith sentence in training data. Let x_{i1} is the parse with the highest f-score among all the parses for the ith sentence.

We may take x_{i1} as positive samples, and $x_{ij(j>1)}$ as negative samples. However, experiments have shown that this is not the best way to utilize SVMs in reranking (Dijkstra, 2001). A trick to be used here is to take a pair of parses as a sample: for any i and $j > 1$, (x_{i1}, x_{ij}) is a positive sample, and (x_{ij}, x_{i1}) is a negative sample.

Similar idea was employed in the early works of parse reranking. In the boosting algorithm of (Collins, 2000), for each sample (parse) x_{ij}, its margin is defined as $F(x_{i1}, \bar{\alpha}) - F(x_{ij}, \bar{\alpha})$, where F is a score function and $\bar{\alpha}$ is the parameter vector. In (Collins and Duffy, 2002), for each offending parse, the parameter vector updating function is in the form of $\mathbf{w} = \mathbf{w} + \mathbf{h}(x_{i1}) - \mathbf{h}(x_{ij})$, where $\mathbf{w}$ is the parameter vector and $\mathbf{h}$ returns the feature vector of a parse. But neither of these two papers used a pair of parses as a sample and defined functions on pairs of parses. Furthermore, the advantage of using difference between parses was not theoretically clarified, which we will describe in the next section.

As far as SVMs are concerned, the use of parses or pairs of parses both maximize the margin between x_{i1} and x_{ij}, but the one using a single parse as a sample needs to satisfy some extra constraints on the selection of decision function. However these constraints are not necessary (see section 3.3). Therefore the use of pairs of parses has both theoretic and practical advantages.

Now we need to define the kernel on pairs of parses. Let $(t_1, t_2), (v_1, v_2)$ are two pairs of parses. Let $\mathbf{K}$ is any kernel function on the space of single parses. The $preference$ kernel $\mathbf{P_K}$ is defined on $\mathbf{K}$ as follows.

$$\mathbf{P_K}((t_1, t_2), (v_1, v_2)) \equiv \mathbf{K}(t_1, v_1) - \mathbf{K}(t_1, v_2) - \mathbf{K}(t_2, v_1) + \mathbf{K}(t_2, v_2) \quad (6)$$

The $preference$ kernel of this form was previously used in the context of ordinal regression in (Herbrich et al., 2000). Then the decision function is

$$\begin{aligned} f((x_j, x_k)) &= \sum_{i=1}^{N_s} \alpha_i y_i \mathbf{P_K}((s_{i1}, s_{i2}), (x_j, x_k)) + b \\ &= b + (\sum_{i=1}^{N_s} \alpha_i y_i (\mathbf{K}(s_{i1}, x_j) - \mathbf{K}(s_{i2}, x_j))) \\ &\quad -(\sum_{i=1}^{N_s} \alpha_i y_i (\mathbf{K}(s_{i1}, x_k) - \mathbf{K}(s_{i2}, x_k))), \end{aligned}$$

where x_j and x_k are two distinct parses of a sentence, (s_{i1}, s_{i2}) is the ith support vector, and N_s is the total number of support vectors.

As we have defined them, the training samples are symmetric with respect to the origin in the space. Therefore, for any hyperplane that does not pass through the origin, we can always find a parallel hyperplane that crosses the origin and makes the margin larger. Hence, the outcome separating hyperplane has to pass through the origin, which means that $b = 0$.

Therefore, for each test parse x, we only need to compute its score as follows.

$$\mathbf{score}(x) = \sum_{i=1}^{N_s} \alpha_i y_i (\mathbf{K}(s_{i1}, x) - \mathbf{K}(s_{i2}, x)), \quad (7)$$

because

$$f((x_j, x_k)) = \mathbf{score}(x_j) - \mathbf{score}(x_k). \quad (8)$$

2.1 Kernels

In (6), the preference kernel $\mathbf{P_K}$ is defined on kernel $\mathbf{K}$. $\mathbf{K}$ can be any possible kernel. We will show that $\mathbf{P_K}$ is well-defined in the next section. In this paper, we consider two kernels for $\mathbf{K}$, the linear kernel and the tree kernel.

In (Collins, 2000), each parse is associated with a set of features. Linear combination of the features is used in the decision function. As far as SVM is concerned, we may encode the features of each parse with a vector. Dot product is used as the kernel $\mathbf{K}$. Let u and v are two parses. The computational complexity of linear kernel $O(|f_u| * |f_v|)$, where $|f_u|$ and $|f_v|$ are the length of the vectors associated with parse u and v respectively. The goodness of the linear kernel is that it runs very fast in the test phase, because coefficients of the support vectors can be combined in advance. For a test parse x, the computational complexity of test is only $O(|f_x|)$, which is independent with the number of the support vectors.

In (Collins and Duffy, 2002), the tree kernel **Tr** is used to count the total number of common sub-trees of two

parse trees. Let u and v be two trees. Because **Tr** can be computed by dynamic programming, the computational complexity of $\mathbf{Tr}(u, v)$ is $O(|u| * |v|)$, where $|u|$ and $|v|$ are the tree sizes of u and v respectively. For a test parse x, the computational complexity of the test is $O(S * |x|)$, where S is the number of support vectors.

3 Justifying the Algorithm

3.1 Justifying the Kernel

Firstly, we show that the preference kernel $\mathbf{P_K}$ defined above is well-defined. Suppose kernel $\mathbf{K}$ is defined on $\mathbf{T} \times \mathbf{T}$. So there exists $\Phi : \mathbf{T} \mapsto \mathbf{H}$, such that $\mathbf{K}(x_1, x_2) = \Phi(x_1) \bullet \Phi(x_2)$ for any $x_1, x_2 \in \mathbf{T}$.

It suffices to show that there exist space $\mathbf{H}'$ and mapping function $\Phi' : \mathbf{T} \times \mathbf{T} \rightarrow \mathbf{H}'$ such that $\mathbf{P_K}((t_1, t_2), (v_1, v_2)) = \Phi'(t_1, t_2) \bullet \Phi'(t_1, t_2)$, where $t_1, t_2, v_1, v_2 \in \mathbf{T}$.

According to the definition of $\mathbf{P_K}$, we have

$$\begin{aligned} & \mathbf{P_K}((t_1, t_2), (v_1, v_2)) \\ = \; & \mathbf{K}(t_1, v_1) - \mathbf{K}(t_1, v_2) - \mathbf{K}(t_2, v_1) + \mathbf{K}(t_2, v_2) \\ = \; & \Phi(t_1) \bullet \Phi(v_1) - \Phi(t_1) \bullet \Phi(v_2) \\ & -\Phi(t_2) \bullet \Phi(v_1) + \Phi(t_2) \bullet \Phi(v_2) \\ = \; & (\Phi(t_1) - \Phi(t_2)) \bullet (\Phi(v_1) - \Phi(v_2)), \end{aligned} \tag{9}$$

Let $\mathbf{H}' = \mathbf{H}$ and $\Phi'(x_1, x_2) = \Phi(x_1) - \Phi(x_2)$. Hence kernel $\mathbf{P_K}$ is well-defined.

3.2 Margin Bound for SVM-based Voting

We will show that the expected error of voting is bounded from above in the PAC framework. The approach used here is analogous to the proof of ordinal regression (Herbrich et al., 2000). The key idea is to show the equivalence of the voting risk and the classification risk.

Let X be the set of all parse trees. For each $x \in X$, let $\bar{x}$ be the best parse for the sentence related to x. Thus the appropriate loss function for the voting problem is as follows.

$$l_{vote}(x, f) \equiv \begin{cases} 1 & \text{if } \quad f(\bar{x}) < f(x) \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

where f is a parse scoring function.

Let $\mathcal{E} = \{(x, \bar{x}) | x \in X\} \cup \{(\bar{x}, x) | x \in X\}$. $\mathcal{E}$ is the space of event of the classification problem, and $\mathbf{Pr}_{\mathcal{E}}((x, \bar{x})) = \mathbf{Pr}_{\mathcal{E}}((\bar{x}, x)) = \frac{1}{2}\mathbf{Pr}_X(x)$. For any parse scoring function f, let $g_f(x_1, x_2) \equiv sgn(f(x_1) - f(x_2))$. For classifier g_f on space $\mathcal{E}$, its loss function is

$$l_{class}(x_1, x_2, g_f) \equiv \begin{cases} 1 & \text{if } \quad x_1 = \bar{x_2} \text{ and } \\ & \quad g_f(x_1, x_2) = -1 \\ 1 & \text{if } \quad x_2 = \bar{x_1} \text{ and } \\ & \quad g_f(x_1, x_2) = +1 \\ 0 & \text{otherwise} \end{cases}$$

Therefore the expected risk $R_{vote}(f)$ for the voting problem is equivalent to the expected risk $R_{class}(g_f)$ for the classification problem.

$$\begin{aligned} & R_{vote}(f) \\ = \; & \mathbf{E}_{x \in X}(l_{vote}(x, f)) \\ = \; & \mathbf{E}_{(x, \bar{x}) \in \mathcal{E}}(l_{vote}(x, f)) + \mathbf{E}_{(\bar{x}, x) \in \mathcal{E}}(l_{vote}(x, f)) \\ = \; & \mathbf{E}_{(x_1, x_2) \in \mathcal{E}}(l_{class}(x_1, x_2, g_f)) \\ = \; & R_{class}(g_f) \end{aligned} \tag{11}$$

However, the definition of space $\mathcal{E}$ violates the *independently and identically distributed* (*iid*) assumption. Parses for the same sentence are not independent. If we suppose that no two pairs of parses come from the same sentence, then the idd assumption holds. In practice, the number of sentences is very large, i.e. more than 30000. So we may use more than one pair of parses of the same sentence and still assume the *idd* property roughly, because for any two arbitrary pairs of parses, 29999 out of 30000, these two samples are independent.

Let $\rho \equiv \min_{i=1..n, j=2..m_i} |f(x_{i1}) - f(x_{ij})| = \min_{i=1..n, j=2..m_i} |g(x_{i1}, x_{ij}) - 0|$. According to (11) and Theorem 1 in section 1.2 we get the following theorem.

Theorem 2 *If g_f makes no error on the training data, with confidence $1 - \delta$*

$$\begin{aligned} R_{vote}(f) &= R_{class}(g_f) \\ &\leq \frac{2}{m}(k \log(\frac{8em}{k}) \log(32m) \\ &\quad + \log(\frac{8m}{\delta})), \end{aligned} \tag{12}$$

where $k = afat(\rho/8), m = \sum_{i=1..n}(m_i - 1)$.

3.3 Justifying Pairwise Samples

An obvious way to use SVM is to use each single parse, instead of a pair of parse trees, as a training sample. Only the best parse of each sentence is regarded as a positive sample, and all the rest are regarded as negative samples. Similar to the pairwise system, it also maximizes the margin between the best parse of a sentence and all incorrect parses of this sentence. Suppose f is the function resulting from the SVM. It requires $y_{ij} f(x_{ij}) > 0$ for each sample (x_{ij}, y_{ij}). However this constraint is not necessary. We only need to guarantee that $f(x_{i1}) > f(x_{ij})$. This is the reason for using pairs of parses as training samples instead of single parses.

We may rewrite the **score** function (7) as follows.

$$\mathbf{score}(x) = \sum_{i,j} c_{i,j} \mathbf{K}(s_{i,j}, x), \tag{13}$$

where i is the index for sentence, j is the index for parse, and $\forall i \sum_j c_{i,j} = 0$.

The format of **score** in (13) is the same as the decision function generated by an SVM trained on the single parses as samples. However, there is a constraint that the sum of the coefficients related to parses of the same sentence is 0. So in this way we decrease the size of hypothesis space based on the prior knowledge that only the different segments of two distinct parses determine which parse is better.

4 Related Work

The use of pairs of parse trees in our model is analogous to the preference relation used in the ordinal regression algorithm (Herbrich et al., 2000). In that paper, pairs of objects have been used as training samples. For example, let $(r_1, r_2, ...r_m)$ be a list of objects in the training data, where r_i ranks ith. Then pairs of objects (r_{i-1}, r_i) are training samples. Preference kernel $\mathbf{P_K}$ in our paper is the same as the preference kernel in (Herbrich et al., 2000) in format.

However, the purpose of our model is different from that of the ordinal regression algorithm. Ordinal regression searches for a regression function for ordinal values, while our algorithm is designed to solve a voting problem. As a result, the two algorithms differ on the definition of the *margin*. In ordinal regression, the *margin* is $\min |\mathbf{f}(r_i) - \mathbf{f}(r_{i-1})|$, where $\mathbf{f}$ is the regression function for ordinal values. In our algorithm, the *margin* is $\min |\mathbf{score}(x_{i1}) - \mathbf{score}(x_{ij})|$.

In (Kudo and Matsumoto, 2001), SVMs have been employed in the NP chunking task, a typical labeling problem. However, they have used a deterministic algorithm for decoding.

In (Collins, 2000), two reranking algorithms were proposed. In both of these two models, the *loss functions* are computed directly on the feature space. All the features are manually defined.

In (Collins and Duffy, 2002), the Voted Perceptron algorithm was used to in parse reranking. It was shown in (Freund and Schapire, 1999; Graepel et al., 2001) that error bound of (voted) Perceptron is related to margins existing in the training data, but these algorithm are not supposed to maximize margins. Variants of the Perceptron algorithm, which are known as Approximate Maximal Margin classifier, such as PAM (Krauth and Mezard, 1987), ALMA (Gentile, 2001) and PAUM (Li et al., 2002), produce decision hyperplanes within ratio of the maximal margin. However, almost all these algorithms are reported to be inferior to SVMs in accuracy, while more efficient in training.

Furthermore, these variants of the Perceptron algorithm take advantage of the large margin existing in the training data. However, in NLP applications, samples are usually inseparable even if the kernel trick is used. SVMs can still be trained to maximize the margin through the method of soft margin.

5 Experiments and Analysis

We use SVM^{light} (Joachims, 1998) as the SVM classifier. The soft margin parameter C is set to its default value in SVM^{light}.

We use the same data set as described in (Collins, 2000; Collins and Duffy, 2002). Section 2-21 of the Penn WSJ Treebank (Marcus et al., 1994) are used as training data, and section 23 is used for final test. The training data contains around 40,000 sentences, each of which has 27 distinct parses on average. Of the 40,000 training sentences, the first 36,000 are used to train SVMs. The remaining 4,000 sentences are used as development data.

The training complexity for SVM^{light} is roughly $O(n^{2.1})$ (Joachims, 1998), where n is the number of the training samples. One solution to the scaling difficulties is to use the Kernel Fisher Discriminant as described in (Salomon et al., 2002). In this paper, we divide training data into slices to speed up training. Each slice contains two pairs of parses from each sentence. Specifically, slice i contains positive samples $((\tilde{p_k}, p_{ki}), +1)$ and negative samples $((p_{ki}, \tilde{p_k}), -1)$, where $\tilde{p_k}$ is the best parse for sentence k, p_{ki} is the parse with the ith highest log-likelihood in all the parses for sentence k and it is not the best parse. There are about 60000 parses in each slice in average. For each slice, we train an SVM. Then results of SVMs are put together with a simple combination. It takes about 2 days to train a slice on a P3 1.13GHz processor.

As a result of this subdivision of the training data into slices, we cannot take advantage of SVM's global optimization ability. This seems to nullify our effort to create this new algorithm. However, our new algorithm is still useful for the following reasons. Firstly, with the improvement in the computing resources, we will be able to use larger slices so as to utilize more global optimization. SVMs are superior to other linear classifiers in theory. On the other hand, the current size of the slice is large enough for other NLP applications like text chunking, although it is not large enough for parse reranking. The last reason is that we have achieved state-of-the-art results even with the sliced data.

We have used both a linear kernel and a tree kernel. For the linear kernel test, we have used the same dataset as that in (Collins, 2000). In this experiment, we first train 22 SVMs on 22 distinct slices. In order to combine those SVMs results, we have tried mapping SVMs' results to probabilities via a Sigmoid as described in (Platt, 1999).

We use the development data to estimate parameter A and B in the Sigmoid

$$P_i(y = 1|f_i) = \frac{1}{1 + Ae^{-f_i B}}, \tag{14}$$

Table 1: Results on section 23 of the WSJ Treebank. LR/LP = labeled recall/precision. CBs = average number of crossing brackets per sentence. 0 CBs, 2 CBs are the percentage of sentences with 0 or ≤ 2 crossing brackets respectively. CO99 = Model 2 of (Collins, 1999). CH00 = (Charniak, 2000). CO00 = (Collins, 2000).

$\leq$40 Words (2245 sentences)					
Model	LR	LP	CBs	0 CBs	2 CBs
CO99	88.5%	88.7%	0.92	66.7%	87.1%
CH00	90.1%	90.1%	0.74	70.1%	89.6%
CO00	90.1%	90.4%	0.73	70.7%	89.6%
SVM	89.9%	90.3%	0.75	71.7%	89.4%
$\leq$100 Words (2416 sentences)					
Model	LR	LP	CBs	0 CBs	2 CBs
CO99	88.1%	88.3%	1.06	64.0%	85.1%
CH00	89.6%	89.5%	0.88	67.6%	87.7%
CO00	89.6%	89.9%	0.87	68.3%	87.7%
SVM	89.4%	89.8%	0.89	69.2%	87.6%

where f_i is the result of the ith SVM. The parse with maximal value of $\prod_i P_i(y=1|f_i)$ is chosen as the topmost parse. Experiments on the development data shows that the result is better if Ae^{-f_iB} is much larger than 1. Therefore

$$\begin{aligned}\prod_{i=1}^{n} P_i(y=1|f_i) &= \prod_{i=1}^{n} \frac{1}{1+Ae^{-f_iB}} \\ &\approx \prod_{i}^{n} \frac{1}{Ae^{-f_iB}} \\ &= A^{-n}e^{(B\sum_{i=1}^{n} f_i)} \qquad (15)\end{aligned}$$

Therefore, we may use $\sum_i f_i$ directly, and there is no need to estimate A and B in (14). Then we combine SVMs' result with the *log*-likelihood generated by the parser (Collins, 1999). Parameter β is used as the weight of the *log*-likelihood. In addition, we find that our SVM has greater labeled precision than labeled recall, which means that the system prefer parses with less brackets. So we take the number of brackets as another feature to be considered, with weight α. α and β are estimated on the development data.

The result is shown in Table 1. The performance of our system matches the results of (Charniak, 2000), but is a little lower than the results of the Boosting system in (Collins, 2000), except that the percentage of sentences with no crossing brackets is 1% higher than that of (Collins, 2000). Since we have to divide data into slices, we cannot take full advantage of the margin maximization.

Figure 1 shows the learning curves. β is used to control the weight of *log*-likelihood given by the parser. The proper value of β depends on the size of training data. The best result does not improve much after combining 7 slices of training data. We think this is due the limitation of local optimization.

Figure 1: Learning curves on the development dataset of (Collins, 2000). X-axis stands for the number of slices to be combined. Y-axis stands for the F-score.

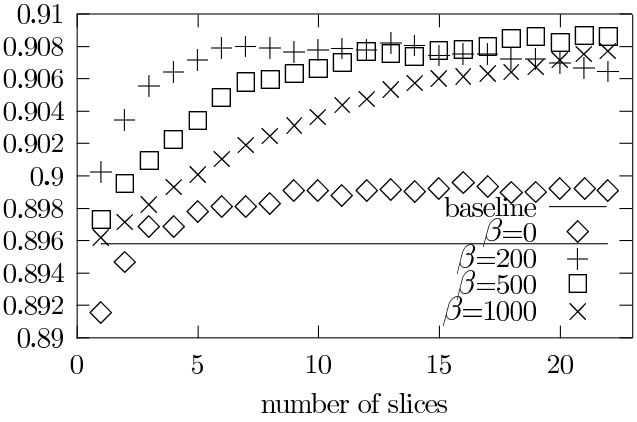

Table 2: Results on section 23 of the WSJ Treebank. LR/LP = labeled recall/precision. CBs = average number of crossing brackets per sentence. CO99 = Model 2 of (Collins, 1999). CD02 = (Collins and Duffy, 2002)

$\leq$100 Words (2416 sentences)			
Model	LR	LP	CBs
CO99	88.1%	88.3%	1.06
CD02	88.6%	88.9%	0.99
SVM(tree)	88.7%	88.8%	0.99

Our next experiment is on the tree kernel as it is used in (Collins and Duffy, 2002). We have only trained 5 slices, since each slice takes about 2 weeks to train on a P3 1.13GHz processor. In addition, the speed of test for the tree kernel is much slower than that for the linear kernel. The experimental result is shown in Table 2 The results of our SVM system match the results of the Voted Perceptron algorithm in (Collins and Duffy, 2002), although only 5 slices, amounting to less than one fourth of the whole training dataset, have been used.

6 Conclusions and Future Work

We have introduced a new approach for applying SVMs to sequential models indirectly, and described a novel SVM based voting algorithm inspired by the parse reranking problem. We have presented a risk formulation under the PAC framework for this voting algorithm, and applied this algorithm to the parse reranking prob-

lem, and achieved LR/LP of $89.4\%/89.8\%$ on WSJ section 23.

Experimental results show that the SVM with a linear kernel is superior to the SVM with tree kernel in both accuracy and speed. The SVM with tree kernel only achieves a rather low f-score because it takes too many unrelated features into account. The linear kernel is defined on the features which are manually selected from a large set of possible features.

As far as context-free grammars are concerned, it will be hard to include more features into the current feature set. If we simply use n-grams on context-free grammars, it is very possible that we will introduce many useless features, which may be harmful as they are in tree kernel systems. One way to include more useful features is to take advantage of the derivation tree and the elementary trees in Lexicalized Tree Adjoining Grammar (LTAG) (Joshi and Schabes, 1997). The basic idea is that each elementary tree and every segment in a derivation tree is linguistically meaningful.

We also plan to apply this algorithm to other sequential models, especially to the Supertagging problem. We believe it will also be very useful to problems of POS tagging and NP chunking. Compared to parse reranking, they have a much smaller training dataset and feature size, which is more suitable for our SVM-based voting problem.

Acknowledgments

We thank Michael Collins for help concerning the data set, and Anoop Sarkar for his comments. We also thank three anonymous reviewers for helpful comments.

References

E. Black, F. Jelinek, J. Lafferty, Magerman D. M., R. Mercer, and S. Roukos. 1993. Towards history-based grammars: Using richer models for probabilistic parsing. In *Proceedings of the ACL 1993*.

Rens Bod. 1998. *Beyond Grammar: An Experience-Based Theory of Language*. CSLI Publications/Cambridge University Press.

E. Charniak. 2000. A maximum-entropy-inspired parser. In *Proceedings of NAACL 2000*.

Michael Collins and Nigel Duffy. 2001. Convolution kernels for natural language. In *Proceedings of Neural Information Processing Systems (NIPS 14)*.

Michael Collins and Nigel Duffy. 2002. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *Proceedings of ACL 2002*.

Michael Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Michael Collins. 2000. Discriminative reranking for natural language parsing. In *Proceedings of the 7th International Conference on Machine Learning*.

N. Cristianini and J. Shawe-Tayor. 2000. *An introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press.

Emma Dijkstra. 2001. Support vector machines for parse selection. Master's thesis, Univ. of Edinburgh.

Yoav Freund and Robert E. Schapire. 1999. Large margin classification using the perceptron algorithm. *Machine Learning*, 37(3):277–296.

Claudio Gentile. 2001. A new approximate maximal margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242.

Thore Graepel, Ralf Herbrich, and Robert C. Williamson. 2001. From margin to sparsity. In *Advances in Neural Information Processing Systems 13*.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. 2000. Large margin rank boundaries for ordinal regression. In *Advances in Large Margin Classifiers*, pages 115–132. MIT Press.

Thorsten Joachims. 1998. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machine*. MIT Press.

A. Joshi and Y. Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69 – 124. Springer.

W. Krauth and M. Mezard. 1987. Learning algorithms with optimal stability in neural networks. *Journal of Physics A*, 20:745–752.

Taku Kudo and Yuji Matsumoto. 2001. Chunking with support vector machines. In *Proceedings of NAACL 2001*.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmentation and labeling sequence data. In *Proceedings of ICML*.

Yaoyong Li, Hugo Zaragoza, Ralf Herbrich, John Shawe-Taylor, and Jaz Kandola. 2002. The perceptron algorithm with uneven margins. In *Proceedings of the International Conference of Machine Learning*.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

John Platt. 1999. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*. MIT Press.

Jesper Salomon, Simon King, and Miles Osborne. 2002. Framewise phone classification using support vector machines. In *Proceedings of ICSLP 2002*.

John Shawe-Taylor, Peter L. Bartlett, Robert C. Williamson, and Martin Anthony. 1998. Structural risk minimization over data-dependent hierarchies. *IEEE Trans. on Information Theory*, 44(5):1926–1940.

A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans. 2000. Introduction to large margin classifiers. In A.J. Smola, P. Bartlett, B. Schölkopf, and C. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 1–26. MIT Press.

Vladimir N. Vapnik. 1998. *Statistical Learning Theory*. John Wiley and Sons, Inc.

Vladimir N. Vapnik. 1999. *The Nature of Statistical Learning Theory*. Springer, 2 edition.