JOURNAL OF COMPUTATIONAL BIOLOGY
Volume 10, Number 2, 2003
© Mary Ann Liebert, Inc.
Pp. 187–213

Protein Family Classification Using Sparse Markov Transducers

ELEAZAR ESKIN,[1] WILLIAM STAFFORD NOBLE,[2,4] and YORAM SINGER[3]

ABSTRACT

We present a method for classifying proteins into families based on short subsequences of amino acids using a new probabilistic model called sparse Markov transducers (SMT). We classify a protein by estimating probability distributions over subsequences of amino acids from the protein. Sparse Markov transducers, similar to probabilistic suffix trees, estimate a probability distribution conditioned on an input sequence. SMTs generalize probabilistic suffix trees by allowing for wild-cards in the conditioning sequences. Since substitutions of amino acids are common in protein families, incorporating wild-cards into the model significantly improves classification performance. We present two models for building protein family classifiers using SMTs. As protein databases become larger, data driven learning algorithms for probabilistic models such as SMTs will require vast amounts of memory. We therefore describe and use efficient data structures to improve the memory usage of SMTs. We evaluate SMTs by building protein family classifiers using the Pfam and SCOP databases and compare our results to previously published results and state-of-the-art protein homology detection methods. SMTs outperform previous probabilistic suffix tree methods and under certain conditions perform comparably to state-of-the-art protein homology methods.

Key words: protein family classification, probabilistic suffix trees, machine learning.

1. INTRODUCTION

As databases of proteins classified into families become increasingly available, and as the number of sequenced proteins grows exponentially, techniques to automatically classify unknown proteins into families become more important. Many approaches have been presented for protein classification. Initially, the approaches examined pairwise similarity (Waterman *et al.*, 1991; Altschul *et al.*, 1990). Other approaches to protein classification are based on creating profiles for protein families (Gribskov *et al.*, 1996), those based on consensus patterns using motifs (Bairoch, 1995; Attwood *et al.*, 1998) and HMM-based (hidden Markov model) approaches (Jaakkola *et al.*, 2000; Eddy, 1995; Baldi *et al.*, 1994).

[1] Department of Computer Science, Columbia University, New York, NY 10027.
[2] Department of Genome Sciences, Health Sciences Center, University of Washington, Seattle, WA 98195.
[3] School of Computer Science and Engineering, Hebrew University, Jerusalem, Israel 91904.
[4] Formerly William Noble Grundy.

Recently, probabilistic suffix trees (PST) have been applied to protein family classification. A PST is a model that predicts the next symbol in a sequence based on the previous symbols (for a formal description see, for instance, Willems *et al.* [1995], Ron *et al.* [1996], Helmbold and Shapire [1997]). These techniques have been shown to be effective in classifying proteins into their appropriate family (Bejerano and Yona, 2001; Apostilico and Bejerano, 2000a). This approach is based on the presence of common short sequences throughout the protein family. These common short sequences, or motifs (Bairoch, 1995), are well understood biologically and have been used effectively in protein classification (Bailey and Gribskov, 1998). PSTs are generative models that induce probabilities over subsequences from a protein by building a probability distribution for an element in the protein sequence using the neighboring elements in the sequence. A PST estimates the conditional probability of each element using the suffix of the input sequence, which is then used to measure how well an unknown sequence fits into that family.

One drawback of probabilistic suffix trees is that they rely on exact matches to the conditional (input) sequences. However, in protein sequences of the same family, substitutions of single amino acids in a sequence are extremely common. For example, two subsequences taken from the *3-hydroxyacyl-CoA dehydrogenase* protein family, *VAVIGSGT* and *VGVLGLGT*, are clearly very similar. However, they have at most only two consecutive matching symbols. If we allowed matching gaps or wild-cards (denoted by $\star$), we notice that they match very closely: $V \star V \star G \star GT$. We would therefore expect that probabilistic suffix trees would perform better if they were able to condition the probabilities on sequences containing wild-cards, i.e., they could ignore or skip some of the symbols in the input sequence.

In this paper we present *sparse Markov transducers* (SMTs), a generalization of probabilistic suffix trees which induce *conditional* probabilities over a sequence that contains wild-cards as described above. Sparse Markov transducers build on previous work on mixtures of probabilistic transducers presented by Willems *et al.* (1995), Singer (1997) and Pereira and Singer (1999). Specifically, they extend previous probabilistic suffix tree models to allow the incorporation of wild-cards into the model. They also provide a simple generalization from a prediction (generative) model to a transduction (discriminative) model which probabilistically maps sequences over an input alphabet to corresponding sequences over an output alphabet. This formalism allows the input alphabet to be different from the output alphabet. We make use of this generalization in our experiments.

We present two methods of building protein family classifiers using sparse Markov transducers. The two models are used to classify unknown proteins into their appropriate families by extracting all subsequences from the protein using a sliding window. For each subsequence, we obtain a probability associated for each family. A score is obtained for each protein and family by computing the normalized sum of logs of the probabilities for each subsequence. In both of these methods, the key to effective protein classification is the estimation of probability distributions conditional on short sequences (in this case, sequences of amino acids). In both methods, SMTs are used to estimate the probability distributions. The first method is a generative model, which builds a prediction model that approximates a probability distribution over a single amino acid conditioned on the sequence of neighboring amino acids. The second method is a discriminative method, which builds estimates of probability distributions over protein families conditional on sequences of amino acids. That is, in the second method we employ Markov transducers as mappings from amino acid sequences to protein families.

We perform experiments over the Pfam database (Sonnhammer *et al.*, 1997) of protein families and build a model for each protein family in the database. We compare our method to the published results from the PST method of Bejerano and Yona (1999). We also perform experiments over the SCOP database and compare our results to state-of-the-art protein homology methods such as BLAST, HMMER, and the Fisher kernel method (Altschul *et al.*, 1990; Eddy, 1995; Jaakkola *et al.*, 2000).

A challenge in using probabilistic modeling methods is that the models they generate tend to be space inefficient. Apostolico and Bejerano (2000b) present an efficient implementation of their PST method. That algorithm brings the memory efficiency close to the theoretical limit so long as a *single* PST or SMT is concerned. However, the algorithm is rather complex, and it is not clear how to generalize the algorithm to the mixtures approach employed in this paper. We therefore present an efficient implementation for sparse Markov transducers incorporating efficient data structures that use lazy evaluation to significantly reduce the memory usage, allowing better models to fit into memory. We show how the efficient data structures allow us to compute better performing models, which would be impossible to compute otherwise.

This paper builds upon that of Eskin *et al.* (2000) where sparse Markov transducers applied to protein classification were originally presented. This paper includes a more complete set of experiments comparing the method to traditional protein homology methods and a complete mathematical description of the model and its learning algorithm. In addition, this paper also describes in depth an extension to compute discriminative models more efficiently.

The organization of the paper is as follows. We first present the formalism of sparse Markov transducers. Then we describe how we use sparse Markov transducers to build models of protein families. We describe in depth a new construction that involves a mixture of many different sparse Markov transducers and discuss its efficient implementation for protein sequences. Finally, we discuss our classification results over the Pfam and SCOP databases. Some of the details of the sparse Markov transducers are technical in nature and are deferred to the appendices. The SMT software is available at *www.cs.columbia.edu/compbio/smt/*.

2. SPARSE MARKOV TRANSDUCERS

Sparse Markov transducers induce a probability distribution of an output symbol conditioned on a sequence of input symbols. In our application, we are specifically concerned with modeling probability distributions of individual amino acids (output symbol) conditioned on their surrounding amino acids (input sequence). We are also interested in the setting where the underlying distribution is conditioned on sequences that contain wild-cards, such as in the case of protein families. We refer to this case as probabilistic estimation over sparse sequences.

To model the probability distribution, we employ Markov transducers. A Markov transducer is defined to be a probability distribution over output symbols conditioned on a finite set of input symbols. A Markov transducer of order L induces a conditional probability distribution of the form

$$P(Y_t|X_tX_{t-1}X_{t-2}X_{t-3}\ldots X_{t-(L-1)}), \tag{1}$$

where X_k are random variables over an input alphabet Σ_{in} and Y_k is a random variable over an output alphabet Σ_{out}. In this form of probability distribution, the output symbol Y_k is conditional on the L previous input symbols. If $Y_t = X_{t+1}$, then the model is a prediction model.[1] In our application, Equation (1) defines the probability distribution conditioned on the context or sequence of neighboring amino acids. As we discuss below, in the context of protein family classification, the conditioning sequences are sequences of amino acids while Y_t is either a single amino acid or the name of the protein family to which the sequence of amino acids belongs.

Markov models provide a systematic way to model probability distributions conditioned on *complete* input sequences. However, many natural sequences, and in particular biological sequences, exhibit a phenomenon known as sparseness where only fractions of the input sequences carry statistically significant information on the output sequences. In order to achieve good probabilistic estimates, we need to cope with sparseness in an algorithmically efficient manner. Specifically, we employ a transduction model that represents parts of the conditioning sequences as wild-cards. We denote a wild-card by the symbol ϕ, which represents a placeholder for an arbitrary input symbol. Put another way, multiple input sequences are mapped to the same conditioning events in all the places where wild-card symbols appear. Similarly, for notational convenience, we use ϕ^n to represent n consecutive wild-cards (i.e., arbitrary sequences of length n) and ϕ^0 as a placeholder representing no wild-cards. Therefore, a sparse Markov transducer is a conditional probability of the form

$$P(Y_t|\phi^{n_1}X_{t_1}\phi^{n_2}X_{t_2}\ldots\phi^{n_k}X_{t_k}), \tag{2}$$

[1] In order to model "look ahead," we can map every element x_t to $x_{t+\Delta t}$ where Δt is a constant value that refers to the number of input symbols that are "looked ahead." Thus, in this case, the conditioning sequence of random variables would be $X_{t+\Delta t}X_{t+\Delta t-1}X_{t+\Delta t-2}\ldots$ The output, Y_t, remains unchanged.

where $t_i = t - (\sum_{j=1}^{i} n_j) - (i-1)$. For instance, if $\Sigma_{in} = \{A, C, T, G\}$ then both *ACTGA* and *ATATA* will be mapped to the same conditioning event $A\phi^3 A$. In fact, there are exactly 4^3 different sequences of length 5 that are mapped to the conditioning $A\phi^3 A$.

Note that a Markov transducer is a special case of a sparse Markov transducer where $n_i = 0$ for all i. The goal of our algorithm is to estimate a conditional probability of this form based on a set of input sequences and their corresponding outputs. However, our task is complicated due to two factors. First, we do not know which positions in the conditioning input sequence should be wild-cards. Second, the positions of the wild-cards change depending on the *context*, or the specific values of the conditioning sequence. This means that the positions of the wild-cards depend on the actual amino acids in the conditional sequence.

We present two approaches for SMT-based protein classification. The first approach is a generative model where for each protein from a family we estimate a distribution over amino acids conditioned on neighboring amino acids. In the generative model, the output (Y_t) is an amino acid and the input sequence ($X_t X_{t-1} X_{t-2} \ldots$) is the neighboring amino acids. Since protein families contain similar subsequences, subsequences from proteins of a given family will fit better into the distribution estimated for the particular family as opposed to distributions estimated using other families. In the second approach, we build a single model for the entire database which maps a sequence of amino acids to the name of the protein family from which the sequence originated. This model estimates the distribution over protein family names (Y_t) conditioned on a sequence of amino acids ($X_t X_{t-1} X_{t-2} \ldots$). In this model, the input alphabet is the set of amino acids, and the output alphabet is the set of protein family names.

In general, our approach is as follows. Building upon the work on suffix trees for learning and prediction (Willems *et al.*, 1995; Ron *et al.*, 1996; Helmhold and Shapire, 1997; Bejerano and Yona, 1999), we define a type of prediction suffix tree called a *sparse prediction tree* which is representationally equivalent to sparse Markov transducers. These trees probabilistically map input strings to a probability distribution over the output symbols. The topology of a tree encodes the positions of the wild-cards in the conditioning sequence of the probability distribution. We estimate the probability distributions of these trees from the set of examples. Since a priori we do not know the positions of the wild-cards, we do not know the best tree topology. For this reason, we use a mixture (weighted sum) of trees and update the weights of each tree based on its performance over the set of examples. We update the trees so that the better-performing trees get larger weights while the worse-performing trees get smaller weights. Thus, the data is used to choose the positions of the wild-cards in the conditioning sequences. We now formally describe sparse prediction trees and discuss the algorithm for updating a mixture of such trees efficiently. The technical details of the algorithm that allow for the *exact* computation of the mixture weights for an exponential number of trees are deferred until Section 5.

2.1. *Sparse prediction trees*

For the simplicity of the derivation of our algorithms, rather than using Markov transducers directly, we employ a tree-based representation which is a specific type of prediction suffix tree called a sparse prediction tree. Sparse prediction trees generalize prediction suffix trees that were described by Ron *et al.* (1996) and used by Bejerano and Yona (1999) in the task of probabilistic modeling of biological sequences. A sparse prediction tree is a rooted tree where each node is either a leaf node or contains one branch labeled with ϕ^n for $n \geq 0$ that forks into a branch for each element in Σ_{in} (each amino acid). Each leaf node of the tree is associated with a probability distribution over the output alphabet, Σ_{out} (amino acids). Figure 1 shows a sparse prediction tree. In this tree, each of the leaf nodes, $u_1, \ldots, u_7$, is associated with a probability distribution. The path from the root node to a leaf node represents the conditioning sequence in the probability distribution. We thus label each node using the path from the root of the tree to the node. Since the path contains the wild-card symbol ϕ, there are multiple strings over Σ_{in} that are mapped to a single node. Put another way, each edge labeled ϕ^n can be traversed by all the $|\Sigma|^n$ different sequences of length n.

A tree associates a probability distribution over output symbols conditioned on the input sequence by following an input sequence from the root node to a leaf node, skipping a symbol in the input sequence for each ϕ along the path. The probability distribution conditioned on an input sequence is the probability distribution associated with the leaf node that corresponds to the input sequence. As described below, the tree is trained with a dataset of input sequences x^t and their corresponding output symbols y_t. The input

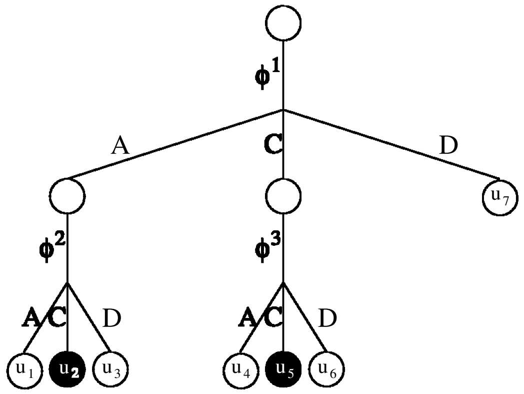

FIG. 1. An illustration of a sparse prediction tree. For space considerations, we do not draw branches for all 20 amino acids.

sequence at time step t, x^t, is the defined to be the entire string of input symbols observed from time step 1 through time step t. Using a notation analogous to Equation (1), let $x_i \in \Sigma_{in}$ be the input symbol that was observed at time t; then $x^t = (x_t, x_{t-1}, \ldots, x_2, x_1)$. Note that this definition implies that the input sequences are strings of growing sizes. In practice, we set a limit on the maximum depth of the sparse prediction trees which implies that each x^t is a finite string whose length is the same as the maximal imposed depth of the learned tree.

For example, in Fig. 1 the sets of input strings that correspond to each of the two highlighted nodes are $u_2 = \phi^1 A \phi^2 C$ and $u_5 = \phi^1 C \phi^3 C$. In our setting, the two nodes would correspond to any amino acid sequences $\star A \star \star C$ and $\star C \star \star \star C$ where the symbol $\star$ denotes a wild-card. The node labeled u_2 in the figure corresponds to many sequences including $A$A$CC$C and $D$A$AC$C. Similarly, the node labeled u_5 in the figure corresponds to sequences ACAAAC and CCADCC. The string CCADCCCA is also mapped to u_5 since the prefix of the sequence leads to u_5. The probability corresponding to an observation from the input sequence is the probability contained in the leaf node corresponding to the conditioning input sequence. For instance, in this example $P(A|AACCC)$ is probability of observing the symbol A associated with the leaf u_2. These probabilities are estimated from counts of output symbols from training examples that reach this node as described below. Note that this is slightly different from the traditional PST formulation (Bejerano and Yona, 1999) where the suffix of a sequence is used to determine the corresponding leaf node. However, these formulations are effectively equivalent because we can preprocess the data to map from one to the other.

In summary, a sparse prediction tree, denoted T, can be used to induce conditional probability distributions over output symbols as follows. For an example pair containing an output symbol y_t and an input sequence x^t, we can determine the conditional probability for the example, denoted $P_T(y_t|x^t)$. As described above, we first determine the node u which corresponds to the input sequence x^t. Once that node is determined, we use the probability distribution over output symbols associated with that node. The prediction of the tree for the example is then $P_T(y_t|x^t) = P_T(y_t|u)$.

We show in Appendix A that any sparse Markov transducer can be represented as a sparse prediction tree of equivalent size. We use, however, the tree-based representation which can naturally be combined with the mixture technique we employ.

2.2. Training a prediction tree

A prediction tree is trained from a set of training examples consisting of output symbols and the corresponding sequences of input symbols. In our application, the training set is either a set of amino acids and their corresponding contexts (neighboring sequence of amino acids) or a set of protein family names and sequences of amino acids from that family. The input symbols are used to identify which leaf node is associated with that training example. The output symbol is then used to update the count of the appropriate predictor.

Each predictor node keeps counts of each output symbol (amino acid) seen by that predictor. We smooth each count by adding a constant value to the count of each output symbol. The predictor's estimate of the probability for a given output is the smoothed count for the output divided by the total count in the predictor.

This method of smoothing is motivated by Bayesian statistics using the Dirichlet distribution which is the conjugate family for the multinomial distribution. However, the discussion of Dirichlet priors is beyond the scope of this paper. Further information on the Dirichlet family can be found in DeGroot (1970). Dirichlet priors have been shown to be effective in protein family modeling (Brown *et al.*, 1995; Sjolander *et al.*, 1996).

For example, consider the prediction tree in Fig. 1. We first initialize all of the predictors (in leaf nodes $u_1, \ldots, u_7$) to the initial count values. If, for example, the first element of training data is the output A and the input sequence *ADCAAACDADCDA*, we would first identify the leaf node that corresponds to the sequence. In this case, the leaf node would be u_7. We then update the predictor in u_7 with the output A by adding 1 to the count of A in u_7. Similarly, if the next output is C and the input sequence is *DACDADDDCCA*, we would update the predictor in u_1 with the output C. If the next output is D and the input sequence is *CAAAACAD*, we would update u_1 with the output D.

After training on these three examples, we can use the tree to output a prediction for an input sequence by using the probability distribution of the node corresponding to the input sequence. For example, assuming the initial count is 0, the prediction of the the input sequence *AACCAAA* which correspond to the node u_1 would give an output probability where the probability for C is .5 and the probability of D is .5.

3. MIXTURE OF SPARSE PREDICTION TREES

In the general case, we do not know a priori where to put the wild-cards in the conditioning sequence of the probability distribution because we do not know on which input symbols the probability distribution is conditional. Thus, we do not know which tree topology to use so as to obtain the best estimate by a sparse prediction tree. Intuitively, we want to use the training data in order to learn which tree predicts most accurately.

We use a Bayesian mixture approach for the problem. Instead of using a single tree as a predictor, we use a mixture technique which employs a weighted sum of trees as our predictor. We then use a Bayesian update procedure to update the weight of each tree based on its performance on each element of the dataset. In this way, the weighted sum uses the data to make the best prediction.

We use a Bayesian mixture for two reasons. First, mixture models provide richer representations than do individual models. Second, our proposed model is built online so that it can be improved on-the-fly with more data, without requiring the retraining of the model. This means that as more and more proteins get classified into families, the models can be updated without retraining the model over the up-to-date database. For a theoretical treatment and further discussion on mixture models of prediction tree see (Willems *et al.*, 1995; Ron *et al.*, 1996; Helmhold and Shapire, 1997) and the references therein.

The skeleton of the learning algorithm for the mixture of SMTs is as follows. We initialize the weights of each SMT in the mixture to the prior probabilities of the trees (discussed below). Then, we update the weight of each tree in an online fashion for each training example in the training set based on how well the tree performed on predicting the most recent output. At the end of this process, we have a weighted sum of (sparse Markov) trees in which the better performing trees have higher weights.

Specifically, we assign a weight, w_T^t, to each tree in the mixture after processing training example t. The prediction of the mixture after training example t is the weighted sum of all the predictions of the trees divided by the sum of all weights:

$$P^t(Y|X^t) = \frac{\sum_T w_T^t P_T(Y|X^t)}{\sum_T w_T^t} \tag{3}$$

where $P_T(Y|X^t)$ is the prediction of tree T for input sequence X^t.

Equation (3) employs a weighted sum of predictions. As discussed above, each weight w_T^t reflects the performance of a tree. Initially, prior to any observations, these weights are initialized in a way that reflects the complexity of each tree in the mixture and are then updated after each round. We now discuss in detail how these mixture weights are determined, starting with how the initial weights are determined.

4. PRIOR DISTRIBUTION OF SPARSE PREDICTION TREES

Our construction of the prior probability of a sparse Markov tree T, denoted w_T^1, is based on the complexity of the topology of the tree. Intuitively, the more complicated the topology of the tree, the smaller its prior probability. We now discuss in detail our construction of a prior probability distribution over SMTs. This construction is recursive and enables us to perform the weight update in an efficient manner. For simplicity of presentation, we describe our construction of a prior probability over SMTs as a stochastic process that generates random trees. The prior probability of a specific tree is the probability of generating that tree according to the process that we now describe.

Given a probability distribution over the (nonnegative) integers, denoted P_ϕ, and starting at the root node, we perform the following process. We pick an integer n at random according to the distribution P_ϕ. If $n = 0$, we stop the generation process making the current node a leaf. Otherwise ($n \geq 1$), we add to the current node a branch labeled ϕ^{n-1} and generate child nodes below that branch, one for every symbol in Σ_{in}. For each of these new nodes, we repeat the process recursively. Clearly, this process induces a (prior) probability of sparse prediction trees.

We refer to the probability distribution induced by the process described above as the generative probability distribution. Intuitively, the outcome of the generation process at each node determines how far forward we look for the next input. If the outcome is 0, then we do not condition on any more inputs. If the value is 1, we condition on the very next input. If the outcome is $n > 2$, then we skip (or mark as wild-cards) the next $n - 1$ inputs and condition on the nth next input.

We can associate a different prior distribution with each node of a sparse prediction tree. Let u be an arbitrary node. We denote by P_ϕ^u the generative probability distribution over the possible skip lengths associated with u. Since P_ϕ^u is a distribution, we have that

$$\sum_{i=0}^{\infty} P_\phi^u(i) = 1. \tag{4}$$

For each node in a tree u, we denote the actual number of skips that were picked according to P_ϕ^u by as u_ϕ. The value $u_\phi - 1$ is the number of ϕ's associated with the (single) edge leaving the node u. Finally, if a node u is a leaf, u_ϕ of that node is defined to be 0.

To complete our description of the prior probability of a tree, we need to introduce a few more definitions. These definitions will also be useful in the weight update procedure for sparse prediction trees. For a tree T, we denote by L_T the set of leaves of that tree. We also define N_T to be the set of nodes of the tree (including the leaf nodes). Let T_u denote the subtree rooted at u. Similarly, we define N_{T_u} and L_{T_u} to be the set of nodes and leaf nodes, respectively, of the subtree T_u.

The prior probability of a tree can now easily be computed using the generative probability distribution at each node and the actual ϕ *value* of each node. Summing up, the prior probability of tree T, denoted w_T^1, is therefore

$$w_T^1 = \prod_{u \in N_T} P_\phi^u(u_\phi) \tag{5}$$

where u_ϕ is the ϕ *value* of the node u and P_ϕ^u is the generative probability distribution associated with the node u.

Assume for example that for all possible nodes $P_\phi^u(n) = \frac{4-n}{10}$ for $0 \leq n \leq 3$ and $P_\phi^u(n) = 0$ otherwise. Figure 2 illustrates the probability of the outcomes (ϕ values) at each node. The prior probability of the entire tree shown in the figure is $(\frac{2}{10})^2(\frac{4}{10})^5 = 0.0004096$, which is the product of the probabilities of each ϕ value at each node.

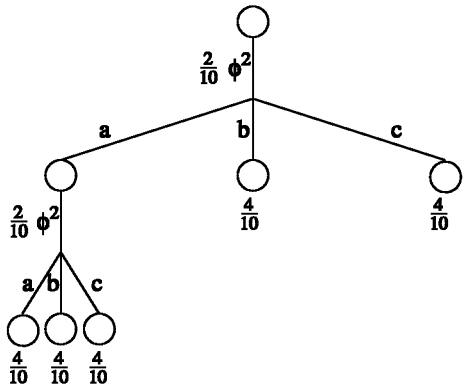

FIG. 2. A sparse prediction tree with its generative probabilities.

Finally, we would like to point out that a maximal depth limit $D_{\max}$ on a sparse prediction tree can be simply imposed by limiting the distributions P^u_ϕ to a finite set of values. Formally, for a node u at depth d, we set $P^u_\phi(n) = 0$ for all values $n > D_{\max} - d$. Note that this implies that for any node u at the maximal depth $D_{\max}$ we have $P^u_\phi(0) = 1$ and $P^u_\phi(n) = 0$ for $n > 0$.

5. WEIGHT UPDATE ALGORITHM

As discussed above, we use a Bayesian approach to update the weights of the mixture after each training example. The mixture weights are updated according to the evidence, which is simply the probability of an output y_t given the input sequence x^t, $P_T(y_t|x^t)$. The prediction is obtained by updating the tree according to the previous example and then computing the prediction of the example. Intuitively, this gives a measure of how well the tree performed on the given example. The unnormalized mixture weights are updated using the following rule:

$$w_T^{t+1} = w_T^t P_T(y_t|x^t) \tag{6}$$

where w_T^1 is defined to be the prior weight of the tree. Thus, the weigh of a tree is the prior weight times the evidence for each training example; that is,

$$w_T^{t+1} = w_T^1 \prod_{i=1}^{t} P_T(y_i|x^i). \tag{7}$$

After each training example we need to update the weights for every possible sparse prediction tree T. Clearly, the number of prediction trees in the mixture is huge, even for relatively small values of $D_{\max}$ and $\phi_{\max}$. Therefore, a straightforward approach that directly updates every sparse prediction tree in the mixture is not feasible. Instead, we present now an efficient algorithm that *tacitly* updates the individual tree weights by maintaining mixture weights of hierarchical subsets of the entire mixture. Our algorithm stems from and builds upon the learning algorithms for prediction suffix trees described by Willems *et al.* (1995) and Helmhold and Shapire (1997).

Efficient weight update

Instead of maintaining multiple sparse prediction trees of different sizes, the efficient algorithm maintains a single template tree that is the union of all the nodes composing the trees in the mixture. At each node of the large template tree, we maintain two weights which we use to update the weights of all the trees in the mixture. These weights are also used for calculating efficiently the mixture's prediction on the next output. We denote the two weights we keep at each node u of the template tree by $w^t(u)$ and $\overline{w}^t(u)$ where t designates the length of the input sequence that has been observed so far.

The first weight $w^t(u)$ is the likelihood of the predictions induced by u on all subsequences reaching the node u. Let us denote by $x^t \in u$ the event that the input sequence x^t has mapped to node u. Recall that there are multiple nodes that x^t is mapped to, including u's ancestors and children. The algorithm updates the weights of all of these nodes in one sweep from the root to all the leaves of the template tree. The weight of each node u is initialized to one, and thus

$$w^1(u) = 1.$$

If $x^t \in u$, then we need to update the likelihood of u as follows:

$$w^{t+1}(u) = w^t(u)P(y_t|u).$$

If, however, $x^t \notin u$, then we need not change the weight of u; therefore, we implicitly set $w^{t+1}(u) = w^t(u)$. Using the weights $w^t(u)$, we can now rewrite the weight of a single tree T in the mixture as follows:

$$w_T^t = w_T^1 \prod_{1 \le i < t} P_T(y_i|x^i) = \left(\prod_{u \in N_T} P_\phi^u(u_\phi) \right) \left(\prod_{v \in L_T} w^t(v) \right). \tag{8}$$

In order to calculate the predictions of the mixture as given by Equation (3), we must keep track of the sum of all the tree weights at time t, $\sum_T w_T^t$. To do so efficiently, we keep track of the sum of all weights for the subtrees rooted at each node. This is the purpose of the second variable $\overline{w}^t(u)$. Let $\mathcal{T}_u$ denote the set of all possible subtrees rooted at u. Then $\overline{w}^t(u)$ is defined to be the sum of all weights of trees in $\mathcal{T}_u$. We calculate $\overline{w}^t(u)$ using Equation (8) as follows:

$$\overline{w}^t(u) = \sum_{T \in \mathcal{T}_u} w_T^t = \sum_{T \in \mathcal{T}_u} \left(\prod_{e \in N_T} P_\phi^e(e_\phi) \right) \left(\prod_{v \in L_T} w^t(v) \right). \tag{9}$$

We now can use the subtree weights to compute the sum of all tree weights in the mixture. Let λ denote the empty string. According to our construction, the set of all prediction trees in the mixture is equal to the set off all subtrees rooted at the root node of the template tree. Therefore, using the the definition of $\mathcal{T}_u$ and the fact that the root node is associated with the empty string, we can rewrite the sum of the weights of all prediction trees in the mixture at time t as

$$\overline{w}^t(\lambda) = \sum_{T \in \mathcal{T}_\lambda} w_T^t = \sum_{T \in \mathcal{T}_\lambda} \left(\prod_{u \in N_T} P_\phi^u(u_\phi) \right) \left(\prod_{v \in L_T} w^t(v) \right). \tag{10}$$

A direct evaluation of the right hand side of Equation (10) is still computationally expensive. However, we can exploit the recursive nature of the the weights to derive an efficient update for $\overline{w}^t(u)$ from the updated weights of all of its children nodes. The following claim gives the form of the update. Its proof is deferred to Appendix B.

Claim 1. *For $t \ge 1$ and for all nodes u in a template sparse prediction tree, the following equality holds:*

$$\overline{w}^t(u) = P_\phi^u(0)w^t(u) + \sum_{i=1}^{\infty} P_\phi^u(i) \prod_{\sigma \in \Sigma_{in}} \overline{w}^t(u\phi^{i-1}\sigma). \tag{11}$$

Equation (11) is the center of the weight update scheme and deserves some further attention. First, note that the equation indeed provides a recursive scheme for computing $\overline{w}^t(u)$ from the weight $w^t(u)$ and the weights of all of u's children, which are of the form $u\phi^{i-1}\sigma$. Though syntactically the recursion includes an infinite sum, the number of the summands is bounded by the minimum between the length of the input sequence so far, which is simply t, and the maximal depth of the prediction trees, $D_{\max}$. Thus, the update of the weights $w^t(u)$ and $\overline{w}^t(u)$ can be done efficiently in one bottom-up pass from the leaf nodes to the root. To summarize, the weight update procedure is as follows:

Initialize: $\overline{w}^1(u)$ for all u in the template tree.
Update $\boldsymbol{w}$: for all u such that $x^t \in u$: $w^{t+1}(u) = w^t(u)P(y_t|u)$.
Update $\overline{\boldsymbol{w}}$: for all u such that $x^t \in u$ $\overline{w}^{t+1}(u) = P^u_\phi(0)w^t(u) + \sum_i P^u_\phi(i)\prod_\sigma \overline{w}^t(u\phi^{i-1}\sigma)$.
Maintain: for all u such that $x^t \notin u$ $w^{t+1}(u) = w^t(u)$; $\overline{w}^{t+1}(u) = \overline{w}^t(u)$.

We note that the above update involves multiple paths. At each node along the path, the input sequence can bifurcate going down to the child node associated with the next input symbol and to all the children associated with wildcards of the form ϕ^i.

In updating our weights, we take advantage of the fact that many of the sequences occur only once in the data. Because of this, many of the subtrees will be traversed by only a single sequence. For the root nodes of subtrees that were traversed only once, we can compute their subtree weight without having to explicitly expand the node into its subtree. Our ability to make this computation stems from the fact that each of the nodes along the path of the sequence will contain only that sequence and thus have the same node weight. More precisely, let u be the root of a subtree that is reached only once. Then, for any node v in the subtree below u, we have that $w^t(u) = w^t(v)$ (for all time steps t). Furthermore, it is simple to verify that the recursive priors imply that for all such nodes u we get $\overline{w}^t(u) = w(u)^t$.

The weight update scheme also serves for outputting predictions. In order to make predictions using the mixtures after training, we can use node weights and subtree weights to compute our predictions efficiently. For any $\hat{y} \in \Sigma_{out}$, the probability of prediction of $\hat{y}$ at time t is

$$P(\hat{y}|x^t) = \frac{\sum\limits_T w^t_T P_T(\hat{y}|x^t)}{\sum\limits_T w^t_T}. \tag{12}$$

If we set $y_t = \hat{y}$, then we have

$$P(y_t|x^t) = \frac{\sum\limits_T w^t_T P_T(y_t|x^t)}{\sum\limits_T w^t_T} = \frac{\sum\limits_T w^{t+1}_T}{\sum\limits_T w^t_T} = \frac{\overline{w}^{t+1}(\lambda)}{\overline{w}^t(\lambda)}. \tag{13}$$

Thus, the prediction of the SMT for an input sequence and output symbol is the ratio of the weight of the root node if the input sequence and output symbol are used to update the tree to the original weight.

When using the mixture of sparse trees for making predictions, we need to compute the probability of all protein families given the conditioning sequence. In other words, we want to compute the probability of every output symbol given the input sequence. The simplest way to do this would be to enumerate over all possible values for y_t (we have $|\Sigma_{out}|$ alternatives) and then compute $\overline{w}^{t+1}$ Σ_{out} using Equation (13). However, this method requires a traversal of the tree $|\Sigma_{out}|$ times, which is time consuming if Σ_{out} is large. Here we can take advantage of the fact that the input sequence is the same for all possible values for y_t and thus the traversed nodes are the same. We can make this computation more efficient if we compute the probabilities for each output symbol in a *single* transversal of the tree. Since the probability for an output symbol is $\frac{\overline{w}^{t+1}}{\overline{w}^t}$ with $\overline{w^{t+1}}$ updated for the output symbol, we compute a vector where each element stores the $\overline{w}^{t+1}$ for the corresponding symbol in Σ_{out}. Now, as we traverse the tree, we update the whole vector, but obtaining the predictions takes only a single sweep through the tree. We can further optimize this scheme since we only need to keep track of the elements in the vector that have been observed in the nodes. That is, the probability of all the unobserved symbols is the same and hence need to be computed only once. These technical improvements significantly reduce the running time when dealing with large alphabets as is the case of the protein families.

6. IMPLEMENTATION ISSUES

In this section we discuss several implementation issues that enable us to cope with relatively large protein datasets. We first describe the constraints that we impose on the structure of SMTs which enable efficient time and space implementation.

The components of the mixture are all possible trees with certain topologies, which can be enormous. We use two parameters to restrict the possible tree topologies in the mixture: $D_{\max}$, the maximum depth of the tree, and $\phi_{\max}$, the maximum number of wild-cards at every node, i.e., the number of consecutive symbols allowed to be skipped. Recall that the depth of a node in the tree is defined to be the length of the input sequence that reaches the node. The maximum number of wild-cards defines the highest power of ϕ on a branch leaving a node. If $\phi_{\max} = 0$, no wild-cards are allowed, and the model reduces to a mixture of prediction suffix trees (PST) where each tree is similar to the form used by Bejerano and Yona (2001). Both $D_{\max}$ and $\phi_{\max}$ affect the number of trees in the mixture. Increasing either value increases the running time of the SMTs and increases the number of total nodes. Even with small values of $D_{\max}$ and $\phi_{\max}$, the number of trees can be very large. For instance, there are ten different trees in the mixture if $D_{\max} = 2$ and $\phi_{\max} = 1$ as shown in Fig. 3 where for illustrative purposes we use an alphabet of size three, $\{A, C, D\}$. With 20 amino acids, there are over a million sparse prediction trees in the mixture.

We can store the set of all trees in the mixture much more efficiently using a *template tree*. This is a single tree that stores the entire mixture. The template tree is similar to a sparse prediction tree except that from each node it has a branch for every possible number of wild-cards at that point in the sequence. A template tree for the trees in the mixture of Fig. 3 is shown in Fig. 4.

Even in the template tree, the maximum number of nodes in the model is also very large. In Fig. 4, there are 16 nodes. However, not every node needs to be stored. We store only the nodes that are reached during training. For example, if the training examples contain the input sequences, AA, AC, and CD, only nine nodes need to be stored in the tree as shown in Fig. 5. This is implemented by starting the algorithm with just a root node and adding elements to the tree as they are reached by examples.

The template tree stores all of the leaf nodes that occur in the trees of the mixture. Each node in the template tree stores a weight of how well that node performs as well as a weight for the subtree rooted

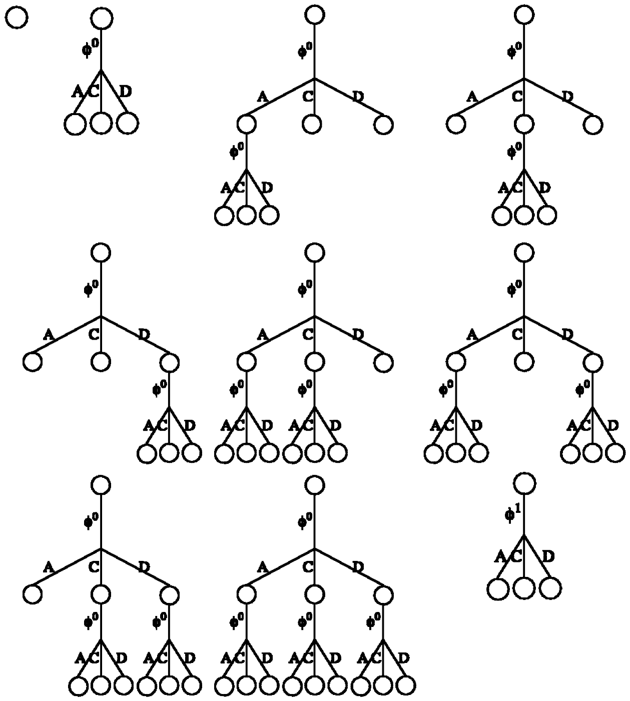

FIG. 3. An illustration of a mixture of sparse suffix trees for $D_{\max} = 2$ and $\phi_{\max} = 1$. In order to simplify the figure, we assume that the input alphabet consists of three symbols $\{A, C, D\}$.

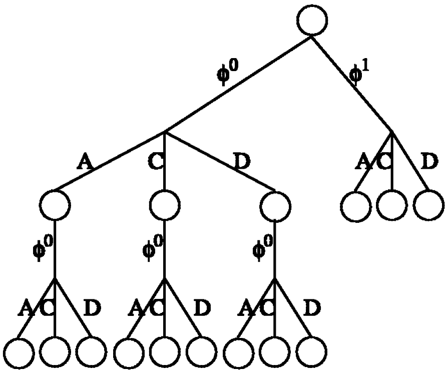

FIG. 4. The template tree for a mixture of sparse prediction trees shown in Fig. 3.

at that node. These weights are efficiently updated during training as discussed in the previous section. Using these weights, we can compute the exact prediction of the mixture as also discussed previously.

Data structures

Even if we only store the nodes reached by the input sequence in the data, the template tree can still grow exponentially fast. With $\phi_{\max} > 0$, the tree branches at each node on every input. Intuitively, this represents the fact that there is an exponential number of possible positions to place the wild-cards in the input sequence. Table 1 shows the number of nodes in a tree with various values of $D_{\max}$ and $\phi_{\max}$ after an empty tree was updated with a single example. Since performance of the SMT typically improves with higher $\phi_{\max}$ and $D_{\max}$, the memory usage becomes a bottleneck because it restricts these parameters to values that will allow the tree to fit in memory and thus the full power of SMTs is not utilized.

Our solution is to use lazy evaluation to provide more efficient data structures with respect to memory. The intuitive idea is that instead of storing all of the nodes created by a training example, we store the tails of the training example (sequence) and recompute part of the tree on demand when necessary. There is an inherent computational cost to this data structure because in many cases the training examples need to be recomputed on demand. Intuitively, we want the parts of the tree that are used often to be stored explicitly as nodes, while the parts of the tree that are not used often are stored as sequences and are recomputed when needed. The data structure is designed to perform exactly the same computation of the SMTs but with significant savings in memory usage. This lazy evaluation approach is rather simple to implement and enables an efficient time and space computation of the prediction of the entire mixture. We would like to note, though, that for a *single* prediction tree there are other approaches that are even more efficient (see Apostilico and Bejerano [2000] and the references therein). However, it is not obvious how to adopt these approaches for our setting which involves *multiple sparse* prediction trees.

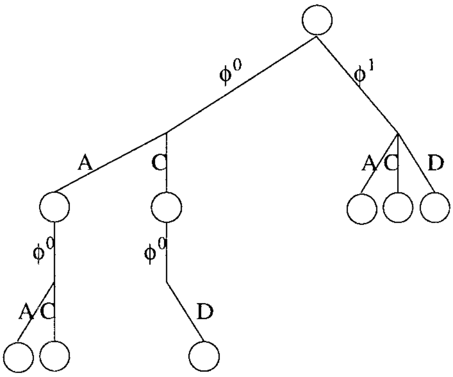

FIG. 5. The template tree of Fig. 4 after processing input sequences *AA*, *AC*, and *CD*.

TABLE 1. NUMBER OF NODES AFTER A SINGLE TRAINING EXAMPLE WITHOUT EFFICIENT DATA STRUCTURES[a]

	ϕmax			
D_{max}	*0*	*1*	*2*	*3*
1	2	2	2	2
2	3	4	4	4
3	4	7	8	8
4	5	12	15	16
5	6	20	28	31
6	7	33	52	60
7	8	54	96	116
8	9	88	177	224
9	10	143	326	432
10	11	232	600	833

[a]The number of nodes generated per examples increases exponentially with ϕ_{max}.

The data structure defines a new way to store the template tree. In this model, the children of nodes in the template tree are either nodes or sequences. Figure 6 gives examples of the data structure. A parameter to the data structure, S_{max}, defines the maximum number of sequences that can be stored on the branch of a node.

Let us look at an example where we are computing a SMT with $D_{\text{max}} = 7$ and $\phi_{\text{max}} = 1$ with the following five input sequences (and the corresponding output symbols in parentheses): *ACDACAC*(*A*), *DACADAC*(*C*), *DACAAAC*(*D*), *ACACDAC*(*A*), and *ADCADAC*(*D*). Without the efficient data structure, the tree contains 241 nodes and takes 31 kilobytes to store. The efficient template tree is shown in Fig. 6a. The efficient data structure contains only one node and ten sequences and takes about 1,000 bytes to store. When $S_{\text{max}} = 3$, each node branch can store up to three sequences before it expands a sequence pointer into a node. In the example shown in Fig. 6a, because there are already three sequences in the branch labeled $\phi^0 A$, any new sequence starting with A will force that branch to expand into a node. Thus, if we add the input sequence *ACDACAC*(*D*), we get the tree in Fig. 6b.

The classification performance of SMTs tends to improve with larger values of D_{max} and ϕ_{max}, as we will show in the results section. The efficient data structures are therefore important since they allow us to compute SMTs with higher values of these parameters.

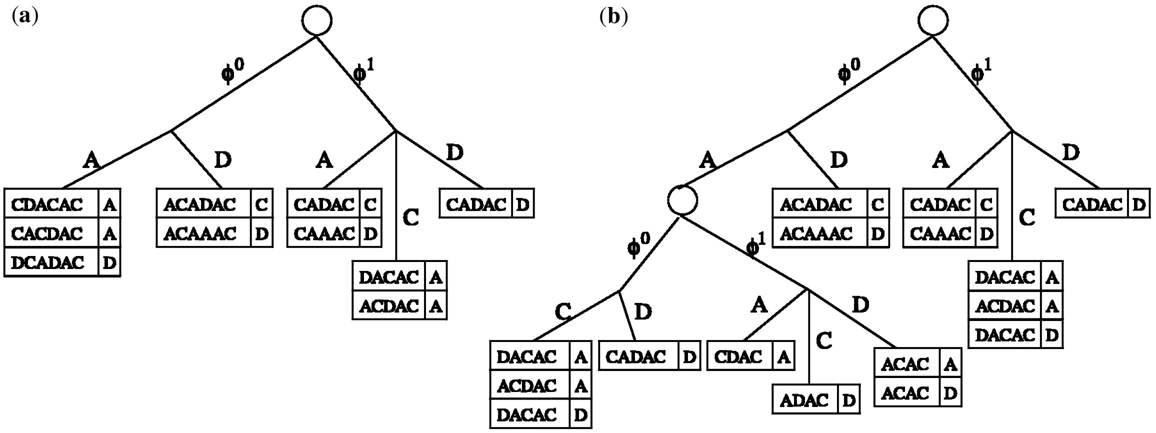

FIG. 6. Efficient data structures for SMTs. The boxes represent input sequences with their corresponding output. (**a**) The tree with $S_{\text{max}} = 3$ after input sequences *ACDACAC*(*A*), *DACADAC*(*C*), *DACAAAC*(*D*), *ACACDAC*(*A*), and *ADCADAC*(*D*). (**b**) The tree after input sequence *ACDACAC*(*D*) is added. Note that a node has been expanded because of the addition of the input.

Short circuit evaluation

We can further optimize the performance by taking advantage of the data being sparse. Since the number of possible sequences seen is significantly smaller than the number of all possible sequences, we can assume that many of the observed sequences will occur only once. This means that many of the nodes in the tree will be reached by only a single sequence. For these nodes, we can compute the subtree weights very efficiently by using the fact that the predictions of all the nodes that have not been reached even once are all the same.

The subtree weight of a node that is only reached by a single sequence can be computed without needing to expand the sequence. In this case, the subtree weight is equivalent to the node, assuming that the predictors in all of the nodes are identical. An explanation of why this is the case was given in Section 5.

This optimization significantly improves performance. When updating a tree with a sequence, we can stop expanding the tree when we reach a node that has not been reached by other sequences. At this point, we can just store the subsequences as described above without having to expand the sequence. Since the data is sparse, this situation occurs in the case of almost every biological sequence.

Coping with skewed distributions

A natural problem to any probability density estimation approach is skew in the distribution of the classes constituting the dataset. For instance, one of the sets of experiments we performed was with the SCOP database. In these experiments, a SMT is trained to recognize the difference between a single family (positive examples) and the remaining families (negative examples). In many of these experiments, the negative examples significantly outnumber the positive examples up to a factor of a hundred to one.

The mixture algorithm tends to concentrate on the component (sparse tree) which performs well on the training set. In the case of a skewed class distribution, the tree containing only the root node will dominate the mixture because in cases of extreme skew as above, the root node will be accurate 99% of the time. In some cases, this will cause the tree containing just the root node to dominate the mixture. In these cases, the prediction of the mixture would be the same regardless of the input sequence. While the empirical distribution of the classes reflects the true distribution, for practical purposes we would like to be able to accurately recognize and retrieve the protein family considered, possibly at the expense of a higher false positive rate. To do so, we experimented with three methods for attempting to compensate for skew in class distribution: leaving out data, replicating data, and weighting data.

The first two approaches are straightforward and involve changing the training data to avoid the problem of skew. The first approach uses only a portion of the negative data. In this case, we keep the amount of positive data fixed. We take a random sample of the negative data so that the numbers of positive and negative examples are equal. In extreme cases, we are using only 1% of the negative data. The second approach was to replicate the positive data in order to have the same amount of positive and negative data. In this approach, we keep the amount of negative data fixed. We replicate the positive data until we have the same amount of total data. In extreme cases, we may have close to hundred copies of the positive data in the training set.

The third approach involves changing the way the predictions are calculated. In this approach, each output symbol has a certain weight corresponding to its ratio in the data. In the case of extreme skew, the positive example will have close to one hundred times the weight that the negative examples have. The node predictors use this weight to update the counts during training. In this case, after training, the root node will contain the same amount of probability mass for each output symbol. The weighted approach is the most efficient and does not dispose of examples and thus was used in our experiments. However, in the cases of extreme skew, even weighting the examples did not completely compensate for the skew, and performance decreases significantly as shown below. This is a known problem in decision theoretic settings, and further investigation of it is beyond the scope of this paper.

7. METHODOLOGY

We use SMTs to perform two sets of experiments. The first set of experiments is using the Pfam database, comparing our results to the results of Bejerano and Yona (1999) over the same data. The second set of

TABLE 2. TIME-SPACE-PERFORMANCE TRADEOFFS FOR THE SMT FAMILY MODEL TRAINED ON THE ABC TRANSPORTERS FAMILY WHICH CONTAINED A TOTAL OF 330 SEQUENCES[a]

			Normal		*Efficient*	
D_{max}	ϕ_{max}	*ROC_{50} score*	*Time*	*Space*	*Time*	*Space*
5	0	.89	1.87	8.6	2.83	2.0
5	1	.90	6.41	30.5	10.57	10.2
5	2	.90	8.55	36.7	13.42	13.6
5	3	.90	9.13	38.8	14.79	14.3
7	0	.89	2.77	17.5	4.22	2.5
7	1	.90	21.78	167.7	37.02	23.3
7	2	.92	37.69	278.1	65.99	49.8
7	3	.92	45.58	321.1	77.47	62.3
9	0	.89	3.69	26.8	6.2	2.9
9	1	.91	—	—	102.35	36.0
9	2	.94	—	—	238.92	108.2
9	3	.93	—	—	324.69	163.7

[a] Time is measured in seconds and space is measured in megabytes. The normal and efficient columns refer to the use of the efficient sequence-based data structures. Because of memory limitations, without using the efficient data structures, many of the models with high values of the parameter values were impossible to compute (indicated with —).

experiments uses the SCOP database (Murzin *et al.*, 1995) and compares the results to state-of-the-art methods in protein classification.

For the first set of experiments, our methodology draws from similar experiments conducted by Bejerano and Yona (1999), in which PSTs were applied to the problem of protein family classification in the Pfam database. We employed two types of protein classifiers based on SMTs and evaluated them by comparing them to the published results of Bejerano and Yona (1999). The first approach builds an SMT model for each protein family where wild-cards are incorporated in the model. We refer to these models as SMT prediction models. The second model is a single SMT-based classifier trained over the entire database that maps sequences to protein family names. We refer to the second model as the SMT classifier model.

In the second set of experiments, our methodology draws from similar experiments conducted by Jaakkola *et al.* (1999), in which a support vector machine was trained over the SCOP database. We compare the results of our experiments to the results of other state-of-the-art protein homology methods using the same data.

In our experiments, we did not perform any tuning of the parameters in order to prevent bias of our results to the specific data. However, in practice, a system for protein classification could improve its performance by tuning parameters to the specifics of the training data, which is a rather simple task when using trained SMTs.

We also performed a set of experiments to test the efficiency of our implementation method (see Table 2). We performed experiments over one protein family to examine the time-space-performance tradeoffs with various restrictions on the topology of the sparse prediction trees ($D_{\max}$, $\phi_{\max}$). We also examined the time-space tradeoffs using the efficient data structures.

8. Pfam EXPERIMENTS

Pfam data

The data examined comes from the Pfam database. We perform our experiments over two versions of the database. To compare our results to the Bejerano and Yona (1999) method, we use the release version 1.0. The data consists of single-domain protein sequences classified into 175 protein families. We use the SPROT33 database and label each protein into the family according to its Pfam labeling. Pfam identifies a number of domains that are present in the sequences. Some proteins sequences have multiple domains.

There are a total of 52,205 proteins of which 15,610 are classified into families. There are a total of 18,531,384 residues in the data.

The sequences for each family were split into training and test data with a ratio of 4:1. For example, the *7 transmembrane receptor* family contains a total of 530 domains spread over 515 protein sequences. The training set contains 412 of the sequences and the test set contains 106 sequences. The 103 sequences of the training set give 158,623 subsequences that are used to train the model.

Building SMT prediction models

A sliding window of size 11 was used over each sequence to obtain a set of subsequences of size 11, $x_1, \ldots, x_{11}$. Using sparse Markov transducers, we built a model that predicted the middle symbol x_6 using the neighboring symbols. The conditional sequence interlaces the five next symbols with the five previous symbols. Specifically, in each training example for the sequence, the output symbol is x_6 and the input symbols are $x_5x_7x_4x_8x_3x_9x_2x_{10}x_1x_{11}$.

A model for each family is built by training over all of the training examples obtained using this method from the protein sequences in the family. The parameters used for building the SMT prediction model are $D_{\max} = 7$ and $\phi_{\max} = 1$. Note these parameters allow up to four wildcards in any sequence of length 7. For evaluating how much the incorporation of wildcards improves performance, we also build SMT prediction models with $D_{\max} = 7$ and $\phi_{\max} = 0$. We also performed experiments with $\phi_{\max} = 2$; however, the results were not significantly different from when $\phi_{\max} = 1$.

Classification of a sequence using a SMT prediction model

We use the family models to compute the likelihood of an unknown sequence fitting into the protein family. First, we convert the amino acids in the sequences into training examples by the method above. The SMT then computes the probability for each training example. We then compute the length-normalized sum of log probabilities for the sequence by dividing the sum by the number of residues in a sequence. This is the likelihood for the sequence to fit into the protein family.

A sequence is classified into a family by computing the likelihood of the fit for the protein family. If the likelihood is above a threshold, then the sequence is classified into the family.

Building the SMT classifier model

The second model we use to classify protein families estimates the probability over protein families given a sequence of amino acids. This model is motivated by biological considerations. Since the protein families are characterized by similar short sequences (motifs) we can map these sequences directly to the protein family that they originated in. This type of model has been proposed for HMMs (Krogh *et al.*, 1994).

Each training example for the SMT Classifier model contains an input sequence which is an amino acid sequence from a protein family and an output symbol which is the protein family name. For example, the *3-hydroxyacyl-CoA dehydrogenase* family contains in one of the proteins a subsequence *VAVIGSGT*. The training example for the SMT would be the sequence of amino acids (*VAVIGSGT*) as the input sequence and the name of the protein as the output symbol (*3-hydroxyacyl-CoA dehydrogenase*).

The training set for the SMT classifier model is the collection of the training sets for each family in the entire Pfam database. We use a sliding window of 10 amino acids, $x_1, \ldots, x_{10}$. In the training example, the output symbol is the name of the protein family. The sequence of input symbols is the 10 amino acids $x_1, \ldots, x_{10}$. Intuitively, this model maps a sequence in a protein family to the name of the family from which the sequence originated. The parameters used for building the model are $D_{\max} = 5$ and $\phi_{\max} = 1$. It took several minutes to train a mixture of sparse prediction trees, and the resulting model occupied 300 megabytes of memory. The reason for the shorter subsequences than those used for the generative models is because the number of total examples when training the discriminative classifier is the total number of subsequences in each training set for each family in the Pfam database. The number of total subsequences in the discriminative training set is 4,770,636 as opposed to 349,634 examples in the largest generative training set, which is for the family *Protein kinase*.

We use the weighted output symbol approach for building our discriminative classifiers. This is because some families have many more instances than other families. This causes the prediction to tend to be biased toward the dominant families. We weigh each instance such that the total sum of the weighted counts at the root node are the same for each family.

Classification of a sequence using an SMT classifier

A protein sequence is classified into a protein family using the complete Pfam model as follows. We use a sliding window of 10 amino acids to compute the set of substrings of the sequence. Each position of the sequence gives us a probability over the 175 families measuring how likely it is that the substring originated from each family. To classify each sequence into a family, we compute for each family the length-normalized sum of the log likelihood of subsequences fitting into the family. This score corresponds to assuming that all subsequences are independent and the prior probability over families is uniform. Our second assumption is consistent with the way we weigh the output symbols to have the same weighted count in the root node. If we were not reweighing our training examples, we would have to take into account the relative frequencies of the output symbols when applying Bayes' rule to make our prediction over the entire sequence.

Pfam results

We compare the performance of the two models examined to the published results for PSTs (Bejerano and Yona, 1999). We train the models over the training set, and we evaluate performance of the model in classifying proteins on the entire database. To compare with published results, we use the equivalence score measure, which is the number of sequences missed when the threshold is set so that the number of false negatives is equal to the number of false positives (Pearson, 1995). We then compute the percentage of sequences from the family recovered by the model and compare this to published results (Bejerano and Yona, 1999).

We evaluate the performance of each model on each protein family separately. We use the model to attempt to distinguish between protein sequences belonging to a family and protein sequences belonging to all other families. Table 3 gives the equivalence scores for the Pfam database version 1.0 and compares them to previously published results. Figure 7 shows scatterplots of the SMT methods versus previously published results.

We compute a two-tailed signed rank test (Salzberg, 1997) to compare the classifiers. The two-tailed signed rank test assigns a p-value to the null hypothesis that the means of the equivalence score distributions of the two classifiers are not equal. As clearly shown in Table 3, both SMT models outperform the PST models. The best performing model is the SMT Classifier, followed by the SMT Prediction model, followed by the PST Prediction model. The signed rank test p-values for the significance between the classifiers are all $< 1\%$. One explanation as to why the SMT Classifier model performed better than the SMT Prediction model is that it is a discriminative model instead of a purely generative model.

TABLE 3. RESULTS OF PFAM PROTEIN CLASSIFICATION USING SMTs[a]

Protein family name	*Protein family size*	*PST equivalence score*	*SMT $\phi_{max} = 0$ equivalence score*	*SMT prediction equivalence score*	*SMT classifier equivalence score*
7tm_1	515	0.93	0.95	0.98	1.00
7tm_2	36	0.94	0.86	0.94	1.00
7tm_3	12	0.83	0.50	1.00	1.00
AAA	66	0.88	0.85	0.89	0.95
ABC_tran	269	0.84	0.84	0.88	0.98
ATP-synt_A	79	0.92	0.91	0.94	0.96
ATP-synt_C	62	0.92	0.85	0.94	0.97

(*continued*)

Table 3. (*Continued*)

Protein family name	*Protein family size*	*PST equivalence score*	*SMT $\phi_{max} = 0$ equivalence score*	*SMT prediction equivalence score*	*SMT classifier equivalence score*
ATP-synt_ab	180	0.97	0.94	0.98	1.00
C2	78	0.92	0.85	0.94	0.97
COX1	80	0.84	0.80	0.85	1.00
COX2	109	0.98	0.94	0.98	0.99
COesterase	60	0.92	0.88	0.93	0.98
Cys-protease	91	0.88	0.87	0.92	0.99
Cys_knot	61	0.93	0.95	0.95	0.96
DAG_PE-bind	68	0.90	0.84	0.88	0.83
DNA_methylase	48	0.83	0.81	0.83	0.96
DNA_pol	46	0.80	0.83	0.87	0.98
E1-E2_ATPase	102	0.93	0.90	0.93	0.99
EGF	169	0.89	0.87	0.91	0.96
FGF	39	0.97	0.92	0.95	1.00
GATase	69	0.88	0.86	0.90	1.00
GTP_EFTU	184	0.92	0.89	0.95	0.99
HLH	133	0.95	0.92	0.94	0.95
HSP20	129	0.95	0.95	0.97	0.98
HSP70	163	0.96	0.94	0.95	0.99
HTH_1	101	0.84	0.85	0.87	0.89
HTH_2	63	0.86	0.83	0.84	0.83
KH-domain	36	0.89	0.80	0.86	0.88
Kunitz_BPTI	55	0.91	0.89	0.93	1.00
MCPsignal	24	0.83	0.71	0.88	0.92
MHC_I	151	0.98	0.99	1.00	0.50
NADHdh	57	0.93	0.89	0.98	1.00
PGK	51	0.94	0.88	0.96	1.00
PH	75	0.93	0.87	0.89	0.82
Pribosyltran	45	0.89	0.71	0.89	1.00
RIP	37	0.95	0.78	0.86	0.97
RuBisCO_large	311	0.99	0.98	0.99	1.00
RuBisCO_small	99	0.97	0.93	0.96	1.00
S12	60	0.97	0.90	0.97	0.98
S4	54	0.93	0.91	0.94	0.98
SH2	128	0.96	0.91	0.96	1.00
SH3	137	0.88	0.83	0.93	0.97
STphosphatase	86	0.94	0.94	0.95	0.99
TGF-beta	79	0.92	0.91	0.94	0.99
TIM	40	0.93	0.85	0.90	1.00
TNFR_c6	29	0.86	0.83	0.90	0.97
UPAR_LY6	14	0.86	0.64	0.86	1.00
Y_phosphatase	92	0.91	0.85	0.88	0.93
Zn_clus	54	0.81	0.80	0.83	0.87
actin	142	0.97	0.96	0.97	1.00
adh_short	180	0.89	0.89	0.91	0.96
adh_zinc	129	0.95	0.88	0.94	0.96
aldedh	69	0.87	0.87	0.93	0.97
alpha-amylase	114	0.88	0.86	0.92	0.98
aminotran	63	0.89	0.86	0.87	0.92
ank	83	0.88	0.86	0.87	0.90
arf	43	0.91	0.91	0.98	0.98
asp	72	0.83	0.85	0.93	0.99

(*continued*)

TABLE 3. (*Continued*)

Protein family name	*Protein family size*	*PST equivalence score*	*SMT $\phi_{max} = 0$ equivalence score*	*SMT prediction equivalence score*	*SMT classifier equivalence score*
bZIP	95	0.90	0.88	0.89	0.93
beta-lactamase	51	0.86	0.82	0.88	0.98
cNMP_binding	42	0.93	0.83	0.93	0.89
cadherin	31	0.87	0.71	0.90	0.96
cellulase	40	0.85	0.85	0.85	0.97
connexin	40	0.97	0.93	0.93	1.00
copper-bind	61	0.95	0.90	0.95	0.98
cpn10	57	0.93	0.89	0.96	0.96
cpn60	84	0.94	0.92	0.93	1.00
crystall	53	0.98	0.87	0.98	1.00
cyclin	80	0.89	0.88	0.91	0.94
cystatin	53	0.93	0.85	0.87	1.00
cytochrome_b_C	130	0.79	0.78	0.88	0.95
cytochrome_b_N	170	0.98	0.95	0.96	1.00
cytochrome_c	175	0.94	0.93	0.94	0.94
dsrm	14	0.86	0.71	0.93	0.92
efhand	320	0.92	0.93	0.94	0.98
enolase	40	1.00	0.90	0.95	1.00
fer2	88	0.94	0.93	0.98	0.99
fer4	152	0.88	0.86	0.91	0.95
fer4_NifH	49	0.96	0.96	0.96	1.00
fibrinogen_C	18	0.78	0.33	0.83	0.87
filament	139	0.96	0.94	0.94	0.99
fn1	15	0.87	0.73	0.87	0.50
fn2	20	0.90	0.80	0.80	1.00
fn3	161	0.86	0.89	0.92	0.90
gln-synt	78	0.94	0.91	0.97	0.99
globin	681	0.98	0.97	0.97	0.99
gluts	144	0.90	0.90	0.93	0.95
gpdh	117	0.97	0.95	0.97	1.00
heme_1	55	0.93	0.78	0.95	1.00
hemopexin	31	0.90	0.84	0.90	1.00
hexapep	45	0.82	0.80	0.89	0.98
histone	178	0.97	0.96	0.98	1.00
homeobox	383	0.93	0.93	0.93	1.00
hormone	111	0.96	0.94	0.95	0.99
hormone_rec	127	0.95	0.94	0.95	0.96
hormone2	73	0.97	0.86	0.97	1.00
hormone3	53	0.91	0.94	0.94	0.98
ig	884	0.94	0.95	0.96	0.99
il8	67	0.94	0.85	0.94	0.99
ins	132	0.98	0.95	0.97	0.99
interferon	47	0.96	0.96	0.98	1.00
kazal	110	0.94	0.94	0.96	1.00
ketoacyl-synt	38	0.82	0.74	0.87	1.00
kringle	38	0.95	0.89	0.89	1.00
laminin_EGF	16	0.81	0.56	0.81	1.00
laminin_G	19	0.90	0.79	0.84	0.88
ldh	90	0.93	0.91	0.93	1.00
ldl_recept_a	31	0.84	0.71	0.87	0.88
ldl_recept_b	14	0.93	0.64	0.93	1.00

(*continued*)

TABLE 3. (*Continued*)

Protein family name	*Protein family size*	*PST equivalence score*	*SMT $\phi_{max} = 0$ equivalence score*	*SMT prediction equivalence score*	*SMT classifier equivalence score*
lectin_c	106	0.87	0.86	0.92	0.91
lectin_legA	43	0.93	0.81	0.88	1.00
lectin_legB	38	0.82	0.61	0.84	1.00
lig_chan	29	0.97	0.79	0.97	0.97
lipase	23	0.87	0.78	0.96	1.00
lipocalin	115	0.94	0.90	0.93	0.97
lys	72	0.99	0.90	0.96	0.99
metalthio	56	1.00	0.96	0.98	0.96
mito_carr	61	0.89	0.92	0.93	0.93
myosin_head	44	0.77	0.68	0.84	1.00
neur	55	0.96	0.93	0.96	1.00
neur_chan	138	0.97	0.94	0.97	0.99
oxidored_fad	101	0.88	0.86	0.91	0.93
oxidored_molyb	35	0.97	0.74	0.91	1.00
oxidored_nitro	75	0.89	0.85	0.92	0.99
p450	204	0.92	0.91	0.95	0.99
peroxidase	55	0.87	0.89	0.95	0.98
phoslip	122	0.97	0.95	0.98	1.00
photoRC	73	0.99	0.97	1.00	1.00
pilin	56	0.89	0.89	0.93	0.96
pkinase	725	0.85	0.91	0.92	1.00
pou	47	0.96	0.89	0.91	0.98
pro_isomerase	50	0.94	0.90	0.92	0.98
pyr_redox	43	0.84	0.79	0.88	0.98
ras	213	0.96	0.92	0.95	1.00
recA	72	0.96	0.94	0.99	1.00
response_reg	128	0.85	0.84	0.87	0.93
rhv	40	0.95	0.85	0.95	1.00
rnaseA	71	0.99	0.89	0.96	0.97
rnaseH	87	0.86	0.89	0.91	0.80
rrm	141	0.84	0.86	0.88	0.91
rvp	82	0.85	0.87	0.90	0.71
rvt	147	0.88	0.90	0.92	0.99
serpin	98	0.91	0.90	0.94	0.99
sigma54	56	0.84	0.80	0.82	1.00
sigma70	61	0.92	0.85	0.89	0.95
sodcu	66	0.92	0.88	0.97	0.98
sodfe	69	0.93	0.96	0.96	1.00
subtilase	82	0.89	0.87	0.90	0.90
sugar_tr	107	0.86	0.87	0.88	0.87
sushi	75	0.89	0.85	0.91	0.98
tRNA-synt_1	35	0.80	0.80	0.83	0.91
tRNA-synt_2	29	0.83	0.83	0.83	0.90
thiolase	25	0.88	0.80	0.92	1.00
thiored	76	0.85	0.83	0.87	0.97
thyroglobulin_1	32	0.91	0.875	0.91	0.94
toxin	172	0.98	0.97	0.98	1.00
trefoil	20	0.85	0.80	0.85	1.00
trypsin	246	0.91	0.90	0.93	1.00
tsp_1	51	0.88	0.86	0.94	0.94
tubulin	196	0.99	0.98	0.99	1.00

(*continued*)

TABLE 3. (*Continued*)

Protein family name	*Protein family size*	*PST equivalence score*	*SMT $\phi_{max} = 0$ equivalence score*	*SMT prediction equivalence score*	*SMT classifier equivalence score*
vwa	29	0.79	0.72	0.83	0.96
vwc	23	0.74	0.70	0.87	1.00
wap	13	0.85	0.54	0.85	0.92
wnt	102	0.94	0.91	0.94	1.00
zf-C2H2	297	0.92	0.89	0.91	0.96
zf-C3HC4	69	0.85	0.86	0.87	0.82
zf-C4	139	0.96	0.92	0.96	0.99
zf-CCHC	105	0.89	0.90	0.91	0.95
zn-protease	148	0.86	0.87	0.88	0.97
zona_pellucida	26	0.89	0.73	0.92	1.00

[a] The equivalence scores are shown for each model for the 170 families with more than 10 sequences in the database. The parameters used to build the models were $D_{\max} = 7$, $\phi_{\max} = 1$ for the SMT prediction models and $D_{\max} = 5$, $\phi_{\max} = 1$ for the SMT classifier model. To measure the improvement from incorporating wildcards, we also build SMT prediction models with $\phi_{\max} = 0$. The two tailed signed rank test assigns a p-value to the null hypothesis that the means of the equivalence score distributions of the two classifiers are not equal. The best performing model is the SMT Classifier, followed by the SMT Prediction model followed by the PST Prediction model. The signed rank test p-values for the significance between the classifiers are all < 1%.

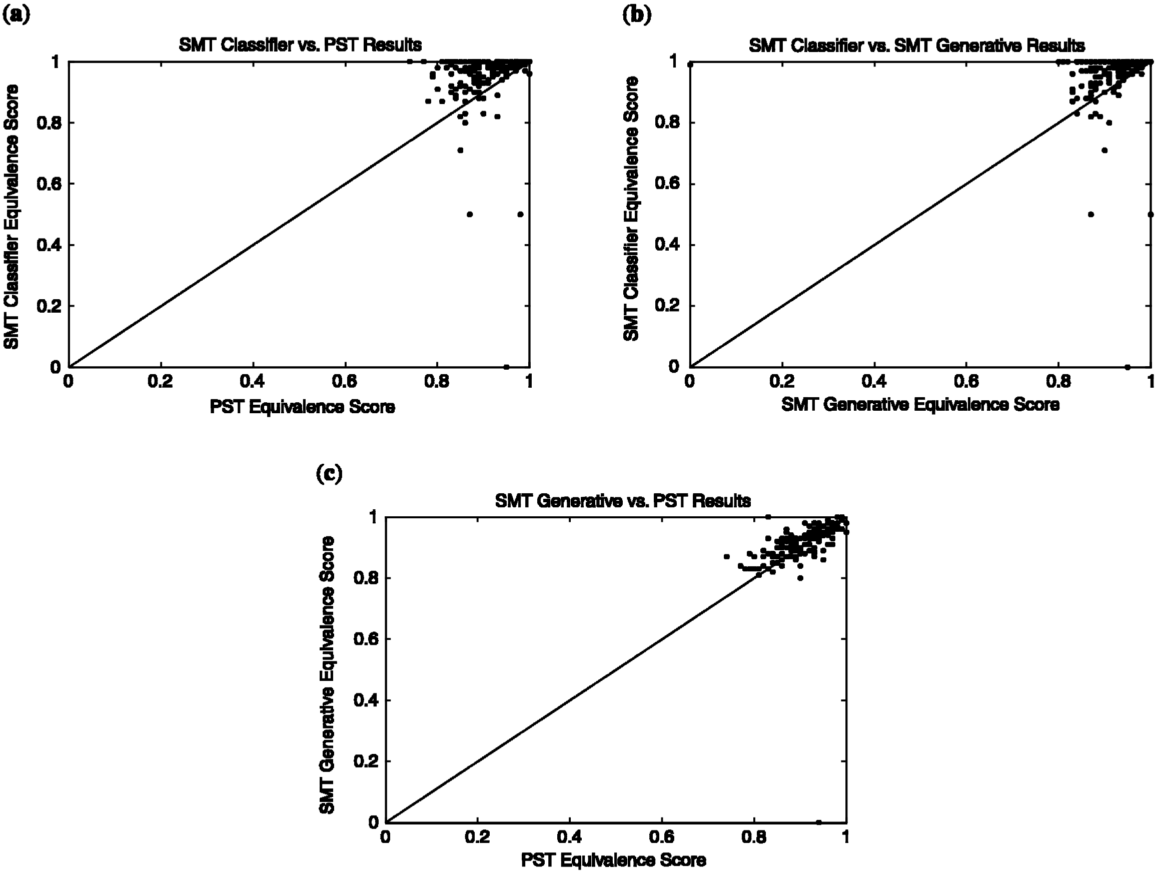

FIG. 7. Scatterplots of equivalence scores from three models of protein classification: (**a**) SMT Classifier versus PST, (**b**) SMT Classifier versus SMT Generative, (**c**) SMT Generative versus PST.

9. SCOP EXPERIMENTS

SCOP data

The second set of experiments were performed over the SCOP database. We used the experimental setup described by Jaakkola *et al.* (1999) and used the data sets obtained from the website *www.cse.ucsc.edu/research/compbio/discriminative/.* We give a brief overview of the data sets below, but more details are available at the web site above.

TABLE 4. PERFORMANCE OF SMT COMPARED TO BLAST, HMMER, AND FISHER KERNEL ON THE SCOP DATABASE[a]

Protein fold	*Target protein family*	*Training protein family*	*Fold*	*Training pos/neg ratio*	*BLAST ROC 50 score*	*HMMER ROC 50 score*	*Fisher-Kernel ROC 50 score*	*SMT ROC 50 score*
2.1	2.1.1.2	2.1.1.4	1	0.79	0.99	0.00	0.97	0.87
2.1	2.1.1.1	2.1.1.4	1	0.79	0.99	0.73	0.99	0.94
2.1	2.1.1.2	2.1.1.4	0	0.73	0.99	0.00	0.97	0.92
2.1	2.1.1.1	2.1.1.4	0	0.73	0.98	0.74	0.99	0.97
1.1	1.1.1.2	1.1.1.1	1	0.47	0.00	0.00	0.00	0.00
1.1	1.1.1.2	1.1.1.1	0	0.44	0.00	0.03	0.00	0.00
1.34	1.34.1.5	1.34.1.4	1	0.40	0.98	1.00	1.00	0.98
1.34	1.34.1.4	1.34.1.5	1	0.39	1.00	1.00	1.00	1.00
1.34	1.34.1.5	1.34.1.4	0	0.37	0.99	1.00	1.00	0.98
1.34	1.34.1.4	1.34.1.5	0	0.36	1.00	1.00	1.00	1.00
1.34	1.34.1.4	1.34.1.1	1	0.26	1.00	1.00	1.00	0.93
1.34	1.34.1.5	1.34.1.2	1	0.25	0.89	1.00	1.00	0.85
1.34	1.34.1.5	1.34.1.1	1	0.25	0.83	1.00	1.00	0.86
1.34	1.34.1.4	1.34.1.2	1	0.25	1.00	1.00	1.00	0.95
1.25	1.25.1.3	1.25.1.1	1	0.24	0.00	0.17	0.52	0.31
1.25	1.25.1.2	1.25.1.1	1	0.24	0.13	0.06	0.53	0.15
1.34	1.34.1.5	1.34.1.1	0	0.23	0.84	1.00	1.00	0.89
1.34	1.34.1.4	1.34.1.1	0	0.23	1.00	1.00	1.00	0.98
1.34	1.34.1.5	1.34.1.2	0	0.22	0.88	1.00	1.00	0.89
1.34	1.34.1.4	1.34.1.2	0	0.22	1.00	1.00	1.00	0.99
1.25	1.25.1.3	1.25.1.1	0	0.21	0.00	0.19	0.55	0.18
1.25	1.25.1.2	1.25.1.1	0	0.21	0.09	0.06	0.61	0.12
2.1	2.1.1.2	2.1.1.5	1	0.12	0.42	0.01	0.11	0.70
2.1	2.1.1.2	2.1.1.1	1	0.11	0.44	0.23	0.47	0.54
2.1	2.1.1.1	2.1.1.5	1	0.10	0.18	0.01	0.12	0.13
2.1	2.1.1.2	2.1.1.5	0	0.08	0.45	0.01	0.28	0.63
2.1	2.1.1.2	2.1.1.1	0	0.08	0.47	0.25	0.52	0.48
2.1	2.1.1.1	2.1.1.5	0	0.07	0.14	0.01	0.20	0.06
1.34	1.34.1.5	1.34.1.3	1	0.03	0.40	0.52	0.71	0.11
1.34	1.34.1.4	1.34.1.3	1	0.03	0.83	0.07	0.99	0.10
1.34	1.34.1.4	1.34.1.3	0	0.03	0.83	0.19	1.00	0.00
1.25	1.25.1.2	1.25.1.3	1	0.03	0.00	0.00	0.35	0.24
1.25	1.25.1.2	1.25.1.3	0	0.03	0.00	0.00	0.42	0.13
1.25	1.25.1.1	1.25.1.3	1	0.03	0.00	0.00	0.00	0.00
1.25	1.25.1.1	1.25.1.3	0	0.03	0.00	0.00	0.09	0.04
1.34	1.34.1.5	1.34.1.3	0	0.02	0.40	0.56	0.66	0.10
1.25	1.25.1.3	1.25.1.2	1	0.02	0.00	0.00	0.15	0.20
1.25	1.25.1.3	1.25.1.2	0	0.02	0.00	0.00	0.28	0.23
1.25	1.25.1.1	1.25.1.2	1	0.02	0.00	0.14	0.17	0.00
1.25	1.25.1.1	1.25.1.2	0	0.02	0.00	0.14	0.23	0.00

[a] The table is sorted with respect to the ration of positive to negative examples in the training set. As expected, for extremely skewed ratios, the SMT performance degrades significantly. The training family is used to seed the HMM which constructs the training set for the experiments. See Jaakkola *et al.* (1999) for a complete discussion.

The data is from the SCOP version 1.37 PDB90 domain database. All SCOP families that contain at least 5 PDB90 sequences and at least 10 PDB90 sequences in the other families in their superfamily were used. This gave 33 test families from 16 superfamilies. Each experiment was created by training on one family in a superfamily and testing the prediction on the remaining sequences in the superfamily. The negative examples are most of the sequences outside of the superfamily. Some sequences outside the superfamily are omitted because of complications described by Jaakkola *et al.* (1999). This gives a total of 160 different training and test sets obtained from the SCOP database. We compare our performance over these 160 tests to other methods on the same data.

Discriminative models over SCOP

In order to build training and test sets for the SMTs, we use an output symbol alphabet of two symbols, $\Sigma_{out} = \{\texttt{positive}, \texttt{negative}\}$. For each of the 160 experiments, we use a sliding window to extract subsequences from the positive and negative training and testing sequences. We label each subsequence positive or negative. We build a SMT for each experiment using the training sequences. We then use the SMT to predict each sequence in the test set by computing the normalized sum of log likelihood of the sequence being in the superfamily (a positive example). This gives a score for each sequence in the test set. We repeat this process for each of the 160 experiments from the SCOP database.

SCOP results

We compare the results of the SMT method against several state-of-the-art protein-classifying methods. We compare against BLAST (Altschul *et al.*, 1990), HMMER (Eddy, 1995), and Fisher kernel based methods (Jaakkola *et al.* 1999). The best performing of these comparison methods is the Fisher kernel based method which is a Support Vector Machine (SVM) approach that uses a generative hidden Markov model as a kernel.

For each of the tests, we report ROC_{50} scores for SMTs and the comparision methods. The ROC_{50} score (Gribskov and Robinson, 1996) is used to evaluate the performance of each method. The ROC_{50} score is the normalized area under the curve that plots true positives versus false positives, up to 50 false positives.

As expected, the performance of the SMT based method degrades with the increase in skew of the data set. We compute the results after applying the three different strategies presented above to handle skew. Table 4 shows the results for the first 40 of the experiments. The table was sorted with decreasing skew to show the effect of skew. We can see that in data sets with low skew, the method performs comparably or even better in some cases to the state-of-the-art methods. However, with higher skew, the performance drops significantly. Note, however, that this skew is an artifact of the specific set of experiments to evaluate these protein homology methods. In practice, the SMT classifier model would be used, which has many classes which prevents the possibility of skew in the data.

In general, SMTs perform worse than the Fisher kernel-based method, but in many cases they perform better than BLAST or HMMER. Note, however, that in some of the "easy" data sets, where HMMER performs perfectly, SMTs do not perform as well as HMMER. This is because HMMER is specifically optimized for these kinds of data sets.

10. EFFICIENCY EXPERIMENTS

We also examined the effect of different parameters on the performance of the model. We examined one of the larger families, *ABC transporters*, containing 330 sequences. Table 2 shows performance of the SMT family model for classifying elements into the *ABC transporters* family as well as the time and space cost of training the model using the two data structures with various settings of the parameters. The efficient data structures allow models with larger parameter values to be computed.

11. DISCUSSION

We have presented two methods for protein classification using sparse Markov transducers (SMTs). The sparse Markov transducers are a generalization of probabilistic suffix trees. The motivation for the sparse

Markov transducers is the presence of common short sequences in protein families. Since substitutions of amino acids are very common in proteins, the models perform more effectively if we model common short subsequences that contain wild-cards. However, it is not clear where to place the wild-cards in the subsequences. The optimal placement of the wild-cards within an amino acid sequence depends on the *context* or neighboring amino acids. We use a mixture technique to learn from the data which placements of wild-cards perform best. We present two models that incorporate SMTs to build a protein classifier. Both of the models out-perform the baseline PST model that does not use wild-cards.

However, the inclusion of wild-cards requires a significant increase in the memory usage of the model. These models can quickly exhaust the available memory. We present efficient data structures that allow for computation of models with wild-cards that otherwise would not be possible. As can be seen in Table 2, without efficient data structures, it would be impossible to compute the models for any but the smallest parameter settings.

A problem with probability density estimators is the problem of skew in the data set. This is a problem that is inherent to probability density estimators and is difficult to fully address. Therefore, over these data sets, it cannot be expected that SMTs perform better than SVM-based methods. However, in practice, SMT-based methods have some advantages over SVM-based methods. Although a single family is relatively small compared to all known proteins, the problem of protein classification is a multiclass problem. In this case, the output symbol alphabet is relatively large, and even the largest protein families comprise only a small part of the overall data. Thus, in the actual application of protein family classification, the problem of skew does not appear. In addition, SMTs have the advantage of being able to build a single model able to discriminate between all protein families. An SVM can distinguish only between two classes so it must build a separate model for each protein family. In the case when there are more than 2,000 protein families this becomes difficult.

The methods presented rely on very little biological intuition. Future work will involve incorporating biological information into the model, such as Dirichlet mixture priors, which can incorporate information about the amino acids (Brown *et al.*, 1995; Sjolander *et al.*, 1996).

APPENDIX

A. Sparse Markov chains as sparse prediction trees

We now show that any sparse Markov transducer can be represented by a prediction tree. The paths of the tree correspond to the conditioning events on the inputs of the sparse Markov transducer. Each sparse Markov transducer of the form given by Equation (2) can be represented with a sparse prediction tree as shown in Fig. 8. Notice that in this tree each of the branches marked ϕ^n attain the same value of n at each level (depth) of the tree. Indeed, all trees that represent conditional probabilities of the form of Equation (2) have this property. In fact, sparse prediction trees can represent a slightly larger class of probability distributions, one that depends on the specific context of the inputs as shown in Fig. 1.

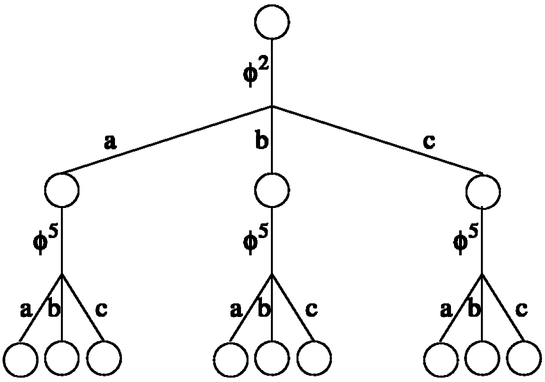

FIG. 8. A sparse prediction tree derived from a sparse Markov chain.

More formally, a fixed order sparse Markov transducer defined by a probability distribution of the form

$$P(Y_t|\phi^{n_1}X_{t-n_1}\phi^{n_2}X_{t-n_1-n_2-1}\ldots\phi^{n_j}X_{t-(L-1)}) \tag{14}$$

can be represented by a sparse prediction tree constructed as follows. We start with just the root node. We add a branch with ϕ^{n_1} and then from this branch a node for every element in Σ_{in}. Then, from each of these nodes, we add a branch with ϕ^{n_2} and then another node for every element in Σ_{in}. We repeat this process, and in the last step we add a branch with ϕ^{n_j} and a node for every element in Σ_{in}. We make these nodes leaf nodes. For each leaf node, u, we associate the probability distribution $P(Y|u)$ determined by the sparse Markov transducer. The probabilistic distribution induced by this tree is equivalent to the probabilistic distribution of the originating sparse Markov transducer.

B. *Proof of Claim 1*

The claim states that for all $t \geq 1$ and all nodes u of a template sparse prediction tree the following holds,

$$\overline{w}^t(u) = P^u_\phi(0)w^t(u) + \sum_{i=1}^{\infty} P^u_\phi(i) \prod_{\sigma\in\Sigma_{in}} \overline{w}^t(u\phi^{i-1}\sigma).$$

First, recall the following definitions. For a tree T, L_T and N_T, respectively, denote the set of leaves and nodes of T. By T_u we denote the subtree rooted at u and similarly N_{T_u} and L_{T_u} were defined to be the set of nodes and leaf nodes of the the subtree T_u.

We now decompose the summation over all subtrees rooted at u with respect to the value of each branch marked ϕ^i. If the i is 0, there is a single tree with only one leaf node which consists of single node u. In this case, the subtree weight is

$$\prod_{v\in N_{T_u}} P^v_\phi(v_\phi) \prod_{v\in L_{T_u}} w^t(v) = P^u_\phi(0)w^t(u). \tag{15}$$

Let us assume that the ϕ *value* of the node u is $i > 0$. In this case, a subtree T_u rooted at u is composed of the node u and a set of subtrees branching off $u\phi^{i-1}\sigma$, for each $\sigma \in \Sigma_{in}$. In accordance with our definition, these subtrees are denoted $T_{u\phi^{i-1}\sigma}$. The set of leaf nodes of the subtree rooted at u will be the union of the leaf nodes of these subtrees. Similarly, the set of nodes of T_u will be the union of the set of nodes of these subtrees and the node u itself. Using this fact, we can represent the weight of each T_u as follows:

$$w^t_{T_u} = P^u_\phi(i) \prod_{\sigma\in\Sigma_{in}} w^t_{T_{u\phi^{i-1}\sigma}}. \tag{16}$$

Let $k = |\Sigma_{in}|$. Then, using the above equation

$$\begin{aligned}\overline{w}^t(u) &= P_\phi(0)w^t(u) + \sum_{i=1}^{\infty} \sum_{T_{u\phi^{i-1}\sigma_1}} \cdots \sum_{T_{u\phi^{i-1}\sigma_k}} P_\phi(i) w^t_{T_{u\phi^{i-1}\sigma_1}} \cdots w^t_{T_{u\phi^{i-1}\sigma_k}} \\ &= P_\phi(0)w^t(u) + \sum_{i=1}^{\infty} P_\phi(i) \prod_{\sigma\in\Sigma_{in}} \sum_{T_{u\phi^{i-1}\sigma}} w^t_{u\phi^{i-1}\sigma},\end{aligned}$$

where we changed the order of summation to get the second inequality. If we now apply the claim recursively to each of the subtrees $T_{u\phi^{i-1}\sigma}$, we get the desired equality,

$$\overline{w}^t(u) = P^u_\phi(0)w^t(u) + \sum_{i=1}^{\infty} P^u_\phi(i) \prod_{\sigma\in\Sigma_{in}} \overline{w}^t(u\phi^{i-1}\sigma).$$

ACKNOWLEDGMENTS

Thanks to Gill Bejerano and Nir Friedman for helpful discussions. The authors thank Mark Diekhans and David Haussler for providing detailed results from their previous experiments. WSN is supported by an Award in Bioinformatics from the PhRMA Foundation and by National Science Foundation grants DBI-0078523 and ISI-0093302. This research of YS was partially funded by the Israeli Science Foundation grant 032.9627.

REFERENCES

Altschul, S.F., Gish, W., Miller, W., Myers, E.W., and Lipman, D.J. 1990. A basic local alignment search tool. *J. Mol. Biol.* 215, 403–410.

Apostolico, A., and Bejerano, G. 2000a. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. *J. Comp. Biol.* 7(3–4), 381–93.

Apostolico, A., and Bejerano, G. 2000b. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. In *Proc. 4th Ann. Int. Conf. Research in Computational Molecular Biology.*

Attwood, T.K., Beck, M.E., Flower, D.R., Scordis, P., and Selley, J.N. 1998. The PRINTS protein fingerprint database in its fifth year. *Nucl. Acids Res.* 26(1), 304–308.

Bailey, T.L., and Gribskov, M. 1998. Methods and statistics for combining motif match scores. *J. Comp. Biol.* 5, 211–221.

Bairoch, A. 1995. The PROSITE database, its status in 1995. *Nucl. Acids Res.* 24, 189–196.

Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M.A. 1994. Hidden Markov models of biological primary sequence information. *Proc. Natl. Acad. Sci. USA* 91(3), 1059–1063.

Bejerano, G., and Yona, G. 1999. Modeling protein families using probabilistic suffix trees. In *Proc. 3rd Ann. Int. Conf. Res. Computational Molecular Biology* 15–24.

Bejerano, G., and Yona, G. 2001. Variations on probabilistic suffix trees—a new tool for statistical modeling and prediction of protein families. *Bioinformatics* 17(1), 23–43.

Brown, M., Hughey, R., Krogh, A., Mian, I., Sjolander, K., and Haussler, D. 1995. Using Dirichlet mixture priors to derive hidden Markov models for protein families. *Proc. 3rd Int. Conf. Intelligent Systems for Molecular Biology*, 47–55.

DeGroot, M.H. 1970. *Optimal Statistical Decisions*, McGraw-Hill, New York.

Eddy, S.R. 1995. Multiple alignment using hidden Markov models. *Proc. 3rd Int. Conf. Intelligent Systems for Molecular Biology* 114–120.

Eskin, E., Grundy, W.N., and Singer, Y. 2000. Protein family classification using sparse markov transducers. *Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology.*

Gribskov, M., Lüthy, R., and Eisenberg, D. 1996. Profile analysis. *Meth. Enzymol.* 183, 146–159.

Gribskov, M., and Robinson, N.L. 1996. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers and Chemistry* 20(1), 25–33.

Helmbold, D.P., and Shapire, R.E. 1997. Predicting nearly as well as the best pruning of a decision tree. *Machine Learning* 27(1), 51–68.

Jaakkola, T., Diekhans, M., and Haussler, D. 1999. Using the Fisher kernel method to detect remote protein homologies. *Proc. 7th Int. Conf. Intelligent Systems for Molecular Biology*, 149–158.

Jaakkola, T., Diekhans, M., and Haussler, D. 2000. A discriminative framework for detecting remote protein homologies. *J. Comp. Biol.* 7, 95–114.

Krogh, A., Brown, M., Mian, I., Sjolander, K., and Haussler, D. 1994. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.* 235, 1501–1531.

Murzin, A.G., Brenner, S.E., Hubbard, T., and Chothia, L. 1995. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.* 247, 536–540.

Pearson, W.R. 1995. Comparison of methods for searching protein sequence databases. *Protein Sci.* 4, 1145–1160.

Pereira, F., and Singer, Y. 1999. An efficient extension to mixture techniques for prediction and decision trees. *Machine Learning* 36(3), 183–199.

Ron, D., Singer, Y., and Tishby, N. 1996. The power of amnesia: Learning probabilistic automata with varibale memory length. *Machine Learning* 25, 117–150.

Salzberg, S.L. 1997. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery* 1, 371–328.

Singer, Y. 1997. Adaptive mixtures of probabilistic transducers. *Neural Computation* 9(8), 1711–1734.

Sjolander, K., Karplus, K., Brown, M., Hughey, R., Krogh, A., Mian, I.S., and Haussler, D. 1996. Dirichlet mixtures: A method for improving detection of weak but significant protein homology. *Comput. Applic. Biosci.* 12(4), 327–345.

Sonnhammer, E., Eddy, S., and Durbin, R. 1997. Pfam: A comprehensive database of protein domain families based on seed alignments. *Proteins* 28(3), 405–420.

Waterman, M.S., Joyce, J., and Eggert, M. 1991. *Phylogenetic Analysis of DNA Sequences*, 59–72, Oxford University Press.

Willems, F.M.J., Shtarkov, Y.M., and Tjalkens, T.J. 1995. The context tree weighting method: Basic properties. *IEEE Transactions on Information Theory* 41(3), 653–664.

Address correspondence to:
Eleazar Eskin
School of Computer Science and Engineering
The Hebrew University
Givat-Ram
Jerusalem, 91904, Israel

E-mail: noble@gs.washington.edu