Protein Family Classification using Sparse Markov Transducers

Eleazar Eskin
Department of Computer Science
Columbia University
eeskin@cs.columbia.edu

William Noble Grundy
Department of Computer Science
Columbia University
bgrundy@cs.columbia.edu

Yoram Singer
School of CSE
Hebrew University
singer@cs.huji.ac.il

Abstract

In this paper we present a method for classifying proteins into families using sparse Markov transducers (SMTs). 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. Because 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. We also present efficient data structures to improve the memory usage of the models. We evaluate SMTs by building protein family classifiers using the Pfam database and compare our results to previously published results.

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, Joyce, & Eggert 1991; Altschul *et al.* 1990). Other approaches to protein classification are based on creating profiles for protein families (Gribskov, Lüthy, & Eisenberg 1990), those based on consensus patterns using motifs (Bairoch 1995; Attwood *et al.* 1998) and HMM-based (hidden Markov model) approaches (Krogh *et al.* 1994; Eddy 1995; Baldi *et al.* 1994).

Recently, probabilistic suffix trees (PSTs) 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 (see for instance (Willems, Shtarkov, & Tjalkens 1995; Ron, Singer, & Tishby 1996; Helmbold & Shapire 1997)). These techniques have been shown to be effective in classifying protein domains and motifs into their appropriate family (Bejerano & Yona 1999; Apostolico & Bejerano 2000). 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 can be used effectively in protein classification (Bailey & Gribskov 1998). These common sequences allow us to build 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 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 only have at most two consecutive matching symbols. If we allowed matching gaps or wild-cards (denoted by $*$), we notice that they match very closely: $V*V*G*GT$. 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 can 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 can condition the probability model over a sequence that contains wild-cards as described above. Sparse Markov transducers build on previous work on mixtures of probabilistic transducers presented in (Willems, Shtarkov, & Tjalkens 1995; Singer 1997; Pereira & Singer 1999). Specifically, they allow the incorporation of wild-cards into the model. They also provide a simple generalization from a prediction model to a transducer model which probabilistically maps any sequence of input symbols to a corresponding output symbol. In a transducer, the input symbol alphabet

Copyright © 2000, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

and output symbol alphabet can be different.

We present two methods of building protein family classifiers using sparse Markov transducers. The first method 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 approximates a probability distribution over the protein families conditional on sequences of amino acids. In the second method we define a mapping from amino acid sequences to protein families. The two models are used to classify unknown proteins into their appropriate families.

We perform experiments over the Pfam database (Sonnhammer, Eddy, & Durbin 1997) of protein families and build a model for each protein family in the database. We compare our method to the Bejerano and Yona (1999) method by comparing the results of our method over the same data to their published results.

The typical problem with probabilistic modeling methods is that the models they generate tend to be space inefficient. Apostolico and Bejerano (2000) present an efficient implementation of their PST method. We 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.

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. Finally, we discuss our classification results over the Pfam database. Since many of the details of the sparse Markov transducers are technical in nature, within the paper we present an outline of the approach with the mathematical details reserved for the appendix.

Sparse Markov Transducers

Sparse Markov transducers estimate a probability distribution of an output symbol (amino acid) conditioned on a sequence of input symbols (sequence of neighboring amino acids). In our application, we are interested in the probability distribution of a single amino acid (output symbols) conditioned on the surrounding amino acids (input sequence). We are interested in the case 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 estimating over sparse sequences.

To model the probability distribution we use Markov transducers. A Markov transducer is defined to be a probability distribution over output symbols conditional over a finite set of input symbols. A Markov transducer of order L is the conditional probability distribution of the form:

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

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 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, (1) defines the probability distribution conditioned on the context or sequence of neighboring amino acids. In the first model (a prediction model), the probability distribution is over amino acids, while in the second model, the probability distribution is over protein families.

In practice, the probability distribution is conditioned on some of the inputs and not the others. We wish to represent a part of the conditional sequence as wild-cards in the probability model. We denote a wild-card with the symbol ϕ, which represents a placeholder for an arbitrary input symbol. Similarly, for notational convenience, we use ϕ^n to represent n consecutive wild-cards and ϕ^0 as a placeholder representing no wild-cards. We use sparse Markov transducers to model this type of distribution. A sparse Markov transducer is a conditional probability of the form:

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

where $t_i = t - (\sum_{j=1}^{i} n_j) - (i-1)$. Note in passing, 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 because of two factors. First, we do not know which positions in the conditional sequence should be wild-cards. Second, the positions of the wild-cards change depending on the *context*, or the specific inputs. 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 prediction model where for each family we estimate a distribution of an amino acid conditioned on its neighboring sequence of amino acids. When modeling prediction probabilities the input and output alphabets are set to be the same. 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 conditioned on a sequence of amino acids. In this model the input alphabet is the set of amino acids and the output alphabet is the set of protein family names.

In brief our approach is as follows. We define a type of prediction suffix tree called a *sparse prediction tree*

[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 which 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.

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. To handle this, we use a mixture (weighted sum) of trees and update the weights of the tree weights based on their 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. In the appendix we present an algorithm for updating the mixture weights and estimating the sparse Markov transducer efficiently. The algorithm allows for the *exact* computation of the mixture weights for an exponential number of trees.

Sparse Markov Trees

To model sparse Markov transducers, we use a type of prediction suffix tree called a sparse prediction tree. 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 Markov tree. In this tree, leaf nodes, $u_1, ...u_7$, each are associated with a probability distribution. The path from the root node to a leaf node represents the conditioning sequence in the probability distribution. We label each node using the path from the root of the tree to the node. Because the path contains the wild-card symbol ϕ, there are multiple strings over Σ_{in} that are mapped to a node. A tree induces a probability distribution over output symbols by following an input string from the root node to a leaf node skipping a symbol in the input string for each ϕ along the path. The probability distribution induced by an input sequence is the probability associated with the leaf node that corresponds to the input sequence. As described later, the tree is trained with a dataset of input sequences x^t and their corresponding output symbols y_t.

For example, in Figure 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 application the two nodes would correspond to any amino acid sequences $*A**C$ and $*C***C$ where $*$ denotes a wild-card. The node labeled u_2 in the figure corresponds to many sequences including $A\mathbf{A}CC\mathbf{C}$ and $B\mathbf{A}AC\mathbf{C}$. Similarly for the node labeled u_5 in the figure corresponds to the sequences $A\mathbf{C}AAA\mathbf{C}$ and $C\mathbf{C}ADC\mathbf{C}$. Also $C\mathbf{C}ADC\mathbf{C}CA$ corresponds to u_5 because the beginning of the sequence corresponds to u_5. The probability corresponding to an input sequence is the probability contained in the leaf node corresponding to the

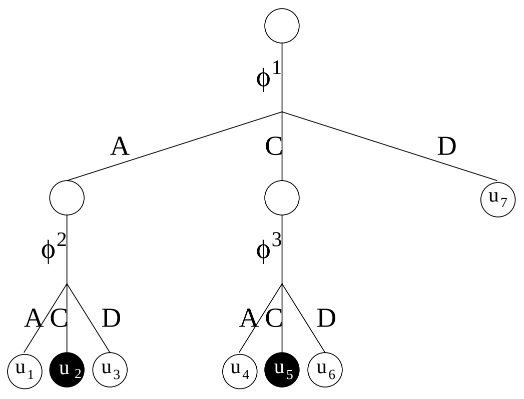

Figure 1. A general sparse Markov tree. For space considerations we do not draw branches for all 20 amino acids.

sequence.

The sparse prediction tree maps sequences to a probability distribution over the output symbols as follows. For a sequence, x^t, we determine the unique leaf node, u, that corresponds to the sequence. The probability over the output symbols is the probability distribution associated with the leaf node.

To summarize, a sparse Markov tree, T, can be used as a conditional probability distribution over output symbols. For a training 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:

$$P_T(y_t|x^t) = P_T(y_t|u) \tag{3}$$

The equivalence between sparse Markov transducers and sparse prediction trees is shown in the appendix.

Training a Prediction Tree

A prediction tree is trained from a set of training examples consisting of an 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.

The predictor kept 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 Figure 1. We first initialize all of the predictors (in leaf nodes $u_1, ..., u_7$) 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 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.

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 can best estimate the distribution. 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.

We use a Bayesian mixture for two reasons. The mixture model provides a richer representation than a single model. Second, the model is built online so that it can be improved on-the-fly with more data, without requiring the re-training of the model. This means that as more and more proteins get classified into families, the models can be updated without re-training the model over the up to date database.

Our algorithm is as follows. We initialize the weights in the mixture to the prior probabilities of the trees. Then we update the weight of each tree for each input string in the dataset based on how well the tree performed on predicting the output. At the end of this process, we have a weighted sum of trees in which the best performing trees in the set of all trees have the highest 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:

$$P^t(Y|X^t) = \frac{\sum_T w_T^t P_T(Y|X^t)}{\sum_T w_T^t} \qquad (4)$$

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

The prior probability of a tree w_T^1, is defined by the topology of the tree. Intuitively, the more complicated the topology of the tree the smaller its prior probability. An effective method to set the prior probabilities is described in the appendix.

General Update Algorithm

We use a Bayesian update rule to update the weights of the mixture for each training example. The prior weight of the tree is w_T^1. 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 with the training example and then computing the prediction of the training 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:

$$w_T^{t+1} = w_T^t P_T(y_t|x^t) \qquad (5)$$

Thus the weigh of a tree is the prior weight times the evidence for each training example:

$$w_T^{t+1} = w_T^1 \prod_{i=1}^{t} P_T(y_i|x^i) \qquad (6)$$

After training example t we update the weights for every tree T. Since the number of possible trees are exponential in terms of the maximum allowed tree depth, this update algorithm requires exponential time.

We present (in the appendix) an update algorithm that computes the exact mixture weights. The efficient algorithm stores and updates weights in the nodes of the tree and uses those weights to compute the mixture of sparse Markov trees. The algorithm for node weight updates does not require exponential time.

Implementation of SMTs

The SMTs can be implemented very efficiently in both time and space.

There are two important parameters for the experiments that define the types of trees in the mixture: MAX_DEPTH, the maximum depth of the tree and MAX_PHI, the maximum number of wild-cards at every node (consecutive wild-cards). 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 MAX_PHI = 0, there are no wild-cards in the model. Both of these variables affect the number of trees in the mixture which increases the running time of the SMTs and increases the number of total nodes.

Even with small values of MAX_DEPTH and MAX_PHI, the number of trees can be very large. For example, there are ten trees in the mixture if MAX_DEPTH= 2 and MAX_PHI= 1 as shown in Figure 2.

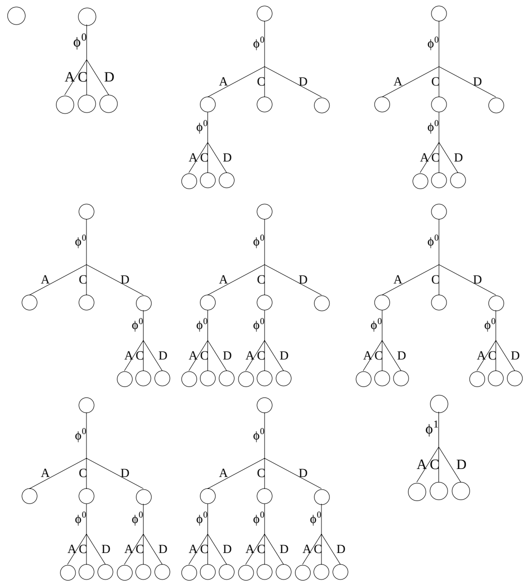

Figure 2. The ten trees in the mixture if MAX_DEPTH= 2 and MAX_PHI= 1. One of the trees consists of only the root node.

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 Figure 2 is shown in Figure 3.

Even in the template tree, the maximum number of nodes in the model is also very large. In Figure 3 there are 16 nodes. However, not every node needs to be stored. We only store these nodes which 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 Figure 4. This is implemented by starting the algorithm with just a root node and adding elements to the tree as they are reached by examples.

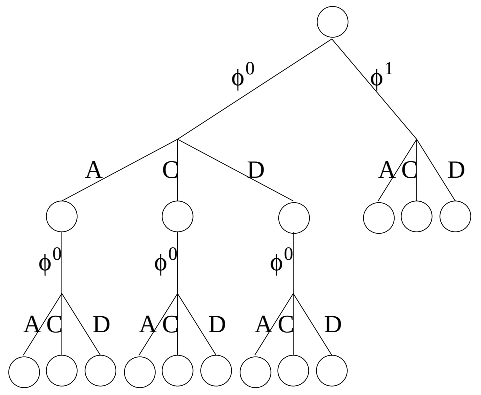

Figure 3. The template tree for the mixture if MAX_DEPTH= 2 and MAX_PHI= 1. For space considerations we do not draw branches for all 20 amino acids.

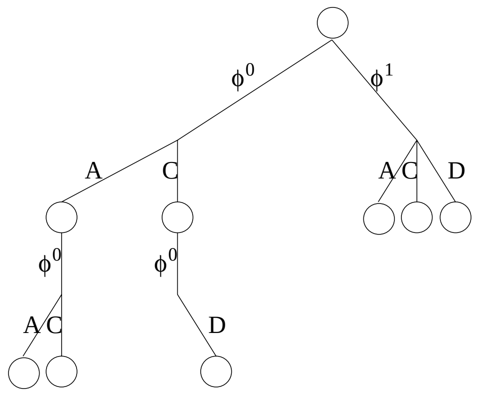

Figure 4. The template tree of Figure 3 after processing input sequences AA, AC and CD.

Efficient Data Structures

Even with only storing the nodes reached by input sequence in the data, the template tree can still grow exponentially fast. With MAX_PHI > 0 the tree branches at each node with every input. Intuitively this represents that fact that there is an exponential number of possible positions of the wild-cards in the input sequence. Table 1 shows the number of nodes in a tree with various values of MAX_DEPTH and MAX_PHI after an empty tree was updated with a single example.

Because performance of the SMT typically improves with higher MAX_PHI nd MAX_DEPTH, the memory usage becomes a bottleneck because it restricts these parameters to values that will allow the tree to fit in memory.

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 the part of

MAX DEPTH	MAX_PHI			
	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

Table 1: Number of nodes after a single training example without efficient data structures. The number of nodes generated per examples increases exponentially with MAX_PHI.

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 to be stored as sequences. The data structure is designed to perform exactly the same computation of the SMTs but with a significant savings in memory usage. We would like to note in passing that there are other approaches that efficiently implement prediction for a single tree (Apostolico & Bejerano 2000).

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 5 gives examples of the data structure. A parameter to the data structure, EXPAND_SEQUENCE_COUNT 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 MAX_DEPTH= 7 and MAX_PHI= 1 with the following 5 input sequences (and corresponding output 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 Figure 5a. The efficient data structure contains only 1 node and 10 sequences and takes 1 kilobyte to store. If the EXPAND_SEQUENCE_COUNT= 4 each node branch can store up to 3 sequences before it expands the sequences into a node. In the example shown in Figure 5a, because there are already 3 sequences in the branch labeled $\phi^0 A$, any new sequence starting with A will force that branch to expand into a node. If we add the input sequence $ACDACAC(D)$ we get the tree in Figure 5b.

The classification performance of SMTs tends to improve with larger values of MAX_DEPTH and MAX_PHI, as we will show in the results section. The efficient data structures are important because they allow us to compute SMTs with higher values of these parameters.

Methodology

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 evaluate two types of protein classifiers based on SMTs and evaluate them by comparing to the published results in (Bejerano & Yona 1999). The first model is a prediction model for each protein family where wildcards 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.

We perform experiments over one protein family to examine the time-space-performance tradeoffs with various restrictions on the topology of the sparse Markov trees (MAX_DEPTH, MAX_PHI). We also examine the time-space tradeoffs using the efficient data structures.

In our experiments, we did not perform any tuning of the parameters.

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. We also compute our models over the latest version of the database, release version 5.2 and results are available online. The data consists of single domain protein sequences classified into 175 protein families. There are a total of 15610 single domain protein sequences containing a total 3560959 residues.

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 protein sequences. The training set contains 424 of the sequences and the test set contains 106 sequences. The 424 sequences of the training set give 108858 subsequences that are used to train the model.

Building SMT Prediction Models

A sliding window of size eleven was used over each sequence to obtain a set of subsequences of size eleven, $a_1, ..., a_{11}$. Using sparse Markov transducers we built a model that predicted the middle symbol a_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 a_6 and the input symbols are $a_5 a_7 a_4 a_8 a_3 a_9 a_2 a_{10} a_1 a_{11}$.

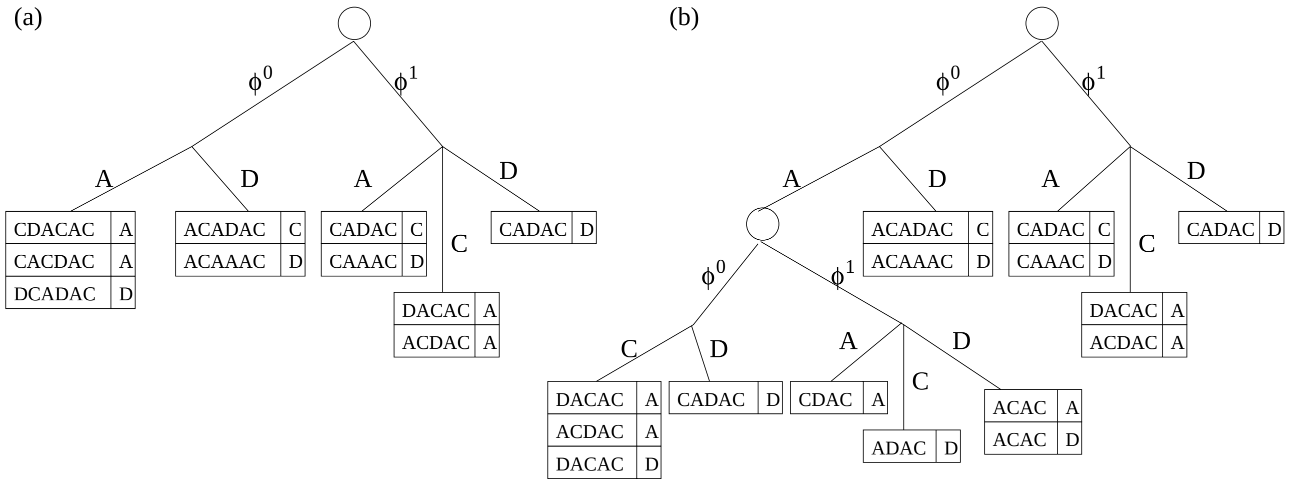

Figure 5. Efficient Data Structures for SMTs. The boxes represent input sequences with their corresponding output. (a) The tree with EXPAND_SEQUENCE_COUNT= 4 after input sequences $ACDACAC(A)$, $DACADAC(C)$, $DACAAAC(D)$, $ACACDAC(A)$ and $ADCADAC(D)$. (b) The tree after input sequence $ACDACAC(D)$ is added. Note that a node has been expanded because of the addition of the input.

A model for each family is built by training over all of the training examples obtained using this method from the amino acid sequences in the family. The parameters used for building the SMT prediction model are MAX_DEPTH= 7 and MAX_PHI= 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 each of the 175 models. Using these likelihoods we can classify a sequence into a family by determining the most likely family for the sequence.

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, $a_1, ..., a_{10}$. In the training example, the output symbol is the name of the protein family. The sequence of input symbols is the 10 amino acids $a_1, ..., a_{10}$. Intuitively, this model maps a sequence in a protein family to the name of the family from where the sequence originated.

The parameters used for building the model are MAX_DEPTH= 5 and MAX_PHI= 1. It took several minutes to train the model and is 300 megabytes in size.

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 the substring originated from each family. To compute the likelihoods of the entire protein sequence fitting into a family, we compute the length normalized sum of log likelihood of subsequences fitting into the family.

Results

We compare the performance of the two models examined to the published results for PSTs (Bejerano & 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 evaluate the performance of a model, we use two measures to evaluate the model predicting a given family. 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 & Yona 1999). We also compute the ROC_{50} score (Gribskov & Robinson 1996). The ROC_{50} score is the normalized area under the curve that plots true positives versus false positives up to 50 false positives.

We evaluated 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 first 50 families in the Pfam database version 1.0 and compares then to previously published results. A complete set of results for the newest database is available online at `http://www.cs.columbia.edu/compbio/smt/`.

We compute a two-tailed signed rank test to compare the classifiers (Salzberg 1997). The two-tailed signed rank test assigns a p-value to the null hypothesis that the means of the two classifiers are not equal. As clearly shown in Table 3, the 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 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.

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.

MAX DEPTH	MAX PHI	ROC_{50} Score	Normal		Efficient	
			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

Table 2: Time-Space-Performance tradeoffs for the SMT family model trained on the ABC transporters family which contained a total of 330 sequences. 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 –).

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 the 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.

The method was evaluated using the Pfam database. The database is biased because it is generated by semi-automatic methods. However, because previous work on PST applied to protein classification used the Pfam database, we used the database for evaluation to compare methods. A better database for evaluation of the model is the SCOP database, which is based on structure of proteins (Murzin *et al.* 1995). Future work involves comparing this method to other protein classification methods on the SCOP database. The method can be evaluated against state of the art methods in protein classification such as the Fisher kernel method (Jaakkola, Diekhans, & Haussler 1999b) and the latest Sam-T HMM methods (Karplus, Barrett, & Hughey 1998).

The methods presented rely on very little biological intuition. Future work involves 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).

Jaakkola shows that combining a generative and discriminative model (such as a support vector machine) can improve performance (Jaakkola, Diekhans, & Haussler 1999a). Another direction for future work involves using this model as a discriminative method that uses both positive and negative examples in training.

Appendix:

Equivalence of sparse Markov chains and sparse prediction trees

Each sparse Markov transducer can be represented by a prediction tree. The paths of the tree correspond to the conditional inputs of the sparse Markov transducer. Each sparse Markov transducer of the form in (2) can be represented with a sparse prediction tree as shown in Figure 6.

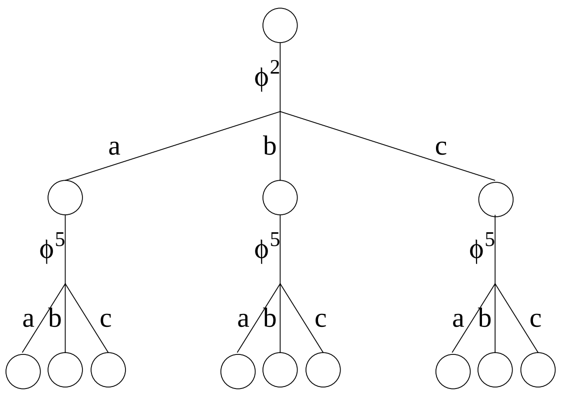

Figure 6. A Sparse Markov Tree

Notice that in this tree each of the ϕ^n have the same value of n at each depth in the tree. The trees that represent conditional probabilities of the form of (2) have this property. In fact, sparse Markov trees can represent a wider class of probability distributions, one that depend on the specific context of the inputs as shown in Figure 1.

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

$$P(Y_t|\phi^{n_1}X_{t-n_1}\phi^{n_2}X_{t-n_1-n_2-1}...\phi^{n_j}X_{t-(L-1)}) \tag{7}$$

can be represented by a sparse Markov 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 mapping of this tree is equivalent to the probabilistic mapping of the sparse Markov transducer.

Prior Probability of a Tree

The initial mixture weights correspond to the prior probabilities of the trees. We define a randomized process that generates sparse prediction trees. The prior probability of a specific tree is the probability of generating that tree.

We define the stochastic process that generates the trees as follows. We start with a single root node. Based on the outcome of a probability distribution over non negative integers, $P_\phi(n \in \mathrm{N})$, we either make this node a leaf node if $n = 0$, or add a branch labeled ϕ^{n-1} and a node for every symbol in Σ_{in} if $n > 0$. For each of these new nodes, we repeat the process recursively. We refer to this probability distribution as the generative probability distribution. Intuitively, this probabilistic event determines how far forward we look for the next input. If the outcome of the probabilistic event is 0, then we do not condition on any more inputs. If the value is 1, we condition on the next input. If the value is $n > 0$, then we skip (or mark as wild-cards) the next $n-1$ inputs and condition on the nth input.

The generative probability distribution $P_\phi()$ is dependent on the current node, u. We will denote this dependence as $P_\phi^u()$. For each node u:

$$\sum_{i=0}^{\infty} P_\phi^u(i) = 1 \tag{8}$$

For each node in a tree u, we denote the outcome of this probabilistic event as u_ϕ which represents the ϕ *value* of that node i.e. the number of ϕ's +1 on the branch leaving the node. If a node is a leaf, u_ϕ of that node is defined to be 0.

For a tree T we define by L_T to be the set of leaves of that tree. We also define N_T to be the set of nodes of the tree. Similarly, we define N_{T_u} and L_{T_u} to be the set of nodes and leaf nodes respectively of a subtree rooted at node u.

The prior probability of a tree can easily be computed using the generative probability distribution at each node and the ϕ *value* of each node. For a tree, T, the prior probability of tree, w_T^1 is then:

$$w_T^1 = \prod_{u \in N_T} P_\phi^u(u_\phi) \tag{9}$$

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

For example, if $P_\phi(n) = \frac{4-n}{10}$ for $0 \leq n \leq 3$ and $P_\phi(n) = 0$ otherwise, Figure 7 shows the generative probability at each node. In this example, the generative probability does not depend on the specific node u. The probability of the tree would be the product of the generative probability at the nodes which is .004096.

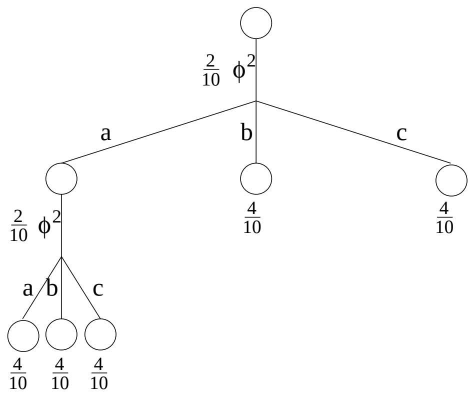

Figure 7. A sparse Markov tree with its generative probabilities.

The initial mixture weights are defined to be these prior probabilities, w_T^1.

The generative probability distribution $P_\phi()$ can be used to define the parameters MAX_PHI and MAX_DEPTH. For example, for a node u with $depth(u)$ =MAX_DEPTH, the node must be a leaf node, thus $P_\phi(0) = 1$.

Weight Update Algorithm

Updating the weights for each tree at every time t is expensive. The update algorithm can be made more efficient if weights are assigned to individual nodes of the template tree which can be updated in a more efficient manner. The mixture weights are then calculated using the node weights.

For each node u we define a weight at time t as follows:

$$w^1(u) = 1 \tag{10}$$

and

$$w^{t+1}(u) = w^t(u)P(y_t|u) \tag{11}$$

when $x^t \in u$ and otherwise $w^{t+1}(u) = w^t(u)$.

Using these weights we can represent the mixture weights.

$$w_T^t = w_T^1 \prod_{1 \leq i < t} P_T(y_i|x^i) = \left(\prod_{u \in N_T} P_\phi^u(u_\phi)\right)\left(\prod_{e \in L_T} w^t(e)\right) \tag{12}$$

In order to make predictions using the mixture (equation (4)), we must keep track of the sum of all the tree weights at time t, $\sum_T w_T^t$. An efficient way to do this is to keep the sum of all subtree weights for each node. We define $\overline{w}^t(u)$ to be the sum of all subtrees rooted at node u:

$$\overline{w}^t(u) = \sum_{T_u}\left(\left(\prod_{e \in N_{T_u}} P_\phi^e(e_\phi)\right)\left(\prod_{v \in L_{T_u}} w^t(v)\right)\right) = \sum_{T_u} w_{T_u}^t \tag{13}$$

We can use these subtree weights to compute the sum of all tree weights, $\sum_T w_T^t$ at time t. Note that the sum of all subtrees rooted at the root node is the sum of all subtrees in the prediction tree:

$$\overline{w}^t(\lambda) = \sum_T\left(\left(\prod_{u \in N_T} P_\phi^u(u_\phi)\right)\left(\prod_{v \in L_T} w^t(v)\right)\right) = \sum_T w_T^t \tag{14}$$

In order to efficiently update the subtree weights we use the following Lemma.

Lemma 1 *The following equation holds:*

$$\overline{w}^t(u) = P_\phi^u(0)w^t(u) + \sum_{i=1}^{\infty} P_\phi^u(i) \prod_{\sigma \in \Sigma_{in}} \overline{w}^t(u\phi^{i-1}\sigma) \tag{15}$$

Proof: We can decompose the summation over all subtrees rooted at u based on the ϕ *value* of the root node u. If the ϕ *value* 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:

$$\prod_{e \in N_{T_u}} P_\phi^e(e_\phi) \prod_{e \in L_{T_u}} w^t(e) = P_\phi^u(0)w^t(u) \tag{16}$$

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 a combination of the node u and a subtree rooted $u\phi^{i-1}\sigma$ for each $\sigma \in \Sigma_{in}$. We denote these subtrees $T_{u\phi^{i-1}\sigma}$. 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 for such T_u:

$$w_{T_u}^t = P_\phi^u(i) \prod_{\sigma \in \Sigma_{in}} w_{T_{u\phi^{i-1}\sigma}}^t \tag{17}$$

Let $k = |\Sigma_{in}|$. Using the above equation:

$$\begin{aligned}\overline{w}^t(u) &= P_\phi(0)w^t(u) \\ &+ \sum_{i=1}^{\infty} \sum_{T_{u\phi^{i-1}\sigma_1}} \dots \sum_{T_{u\phi^{i-1}\sigma_k}} P_\phi(i) w_{T_{u\phi^{i-1}\sigma_1}}^t \dots w_{T_{u\phi^{i-1}\sigma_k}}^t \\ &= P_\phi(0)w^t(u) + \sum_{i=1}^{\infty} P_\phi(i) \prod_{\sigma \in \Sigma_{in}} \sum_{T_{u\phi^{i-1}\sigma}} w_{u\phi^{i-1}\sigma}^t\end{aligned} \tag{18}$$

Thus

$$\overline{w}^t(u) = P_\phi^u(0)w^t(u) + \sum_{i=1}^{\infty} P_\phi^u(i) \prod_{\sigma \in \Sigma_{in}} \overline{w}^t(u\phi^{i-1}\sigma) \tag{19}$$

Efficient Weight Update Rules

To update the weights of the nodes we use the following rules. We first initialize $w^1(u) = 1$ for $\forall u$ and $\overline{w}^1(u)$ for $\forall u$.

For $w^t(u)$ if $x^t \in u$:

$$w^{t+1}(u) = w^t(u)P(y_t|u) \tag{20}$$

and otherwise:

$$w^{t+1}(u) = w^t(u) \tag{21}$$

For $\overline{w}^t(u)$ if $x^t \in u$:

$$\begin{aligned} \overline{w}^{t+1}(u) = \quad & P^u_\phi(0) w^{t+1}(u) \\ + \quad & \sum_{i=1}^{\infty} P^u_\phi(i) \prod_{\sigma \in \Sigma_{in}} \overline{w}^{t+1}(u\phi^{i-1}\sigma) \end{aligned} \tag{22}$$

and otherwise:

$$\overline{w}^{t+1}(u) = \overline{w}^t(u) \tag{23}$$

Notice that each input string x^t corresponds to many nodes u because of the ϕ symbols in the path of u.

Prediction

We can use node weights for efficiently computing the prediction of the mixture. For any $\hat{y} \in \Sigma_{out}$, the probability of prediction of $\hat{y}$ at time t is:

$$P(\hat{y}|x^t) = \frac{\sum_T w^t_T P_T(\hat{y}|x^t)}{\sum_T w^t_T} \tag{24}$$

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

$$P(\hat{y}|x^t) = \frac{\sum_T w^t_T P_T(y_t|x^t)}{\sum_T w^t_T} = \frac{\sum_T w^{t+1}_T}{\sum_T w^t_T} = \frac{\overline{w}^{t+1}(\lambda)}{\overline{w}^t(\lambda)} \tag{25}$$

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.

Acknowledgements

Thanks to G. Bejerano for helpful discussions.

References

Altschul, S. F.; Gish, W.; Miller, W.; Myers, E. W.; and Lipman, D. J. 1990. A basic local alignment search tool. *Journal of Molecular Biology* 215:403–410.

Apostolico, A., and Bejerano, G. 2000. Optimal amnesic probabilistic automata or how to learn and classify proteins in linear time and space. In *Proceedings of RECOMB2000.*

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. *Nucleic Acids Research* 26(1):304–308.

Protein Family Name	Protein Family Size	PST Equiv. Score	SMT Prediction Equiv. Score	SMT Classifier Equiv. Score
7tm_1	515	93.0%	96.2%	97.0%
7tm_2	36	94.4%	97.2%	97.2%
7tm_3	12	83.3%	91.7%	100.0%
AAA	66	87.9%	92.4%	94.9%
ABC_tran	269	83.6%	91.2%	93.3%
ATP-synt_A	79	92.4%	93.7%	94.9%
ATP-synt_C	62	91.9%	96.8%	96.8%
ATP-synt_ab	180	96.7%	98.4%	100.0%
C2	78	92.3%	94.1%	96.0%
COX1	80	83.8%	95.0%	97.5%
COX2	109	98.2%	93.0%	95.6%
COesterase	60	91.7%	88.7%	90.3%
Cys_knot	61	93.4%	96.8%	100.0%
Cys-protease	91	87.9%	93.4%	95.1%
DAG_PE-bind	68	89.7%	93.5%	95.4%
DNA_methylase	48	83.3%	89.5%	91.2%
DNA_pol	46	80.4%	86.3%	88.2%
E1-E2_ATPase	102	93.1%	93.2%	94.0%
EGF	169	89.3%	97.5%	98.8%
FGF	39	97.4%	100.0%	100.0%
GATase	69	88.4%	94.2%	94.2%
GTP_EFTU	184	91.8%	96.7%	98.4%
HLH	133	94.7%	94.0%	98.5%
HSP20	129	94.6%	94.7%	96.2%
HSP70	163	95.7%	98.2%	98.2%
HTH_1	101	84.2%	79.2%	85.1%
HTH_2	63	85.7%	80.0%	81.5%
KH-domain	36	88.9%	88.0%	84.0%
Kunitz_BPTI	55	90.9%	88.5%	92.3%
MCPsignal	24	83.3%	100.0%	100.0%
MHC_I	151	98.0%	100.0%	100.0%
NADHdh	57	93.0%	95.1%	98.4%
PGK	51	94.1%	96.1%	98.0%
PH	75	93.3%	79.2%	83.1%
Pribosyltran	45	88.9%	88.9%	95.6%
RIP	37	94.6%	91.9%	91.9%
RuBisCO_large	311	98.7%	99.7%	99.7%
RuBisCO_small	99	97.0%	98.1%	99.1%
S12	60	96.7%	98.3%	100.0%
S4	54	92.6%	94.4%	96.3%
SH2	128	96.1%	98.0%	98.7%
SH3	137	88.3%	93.2%	96.9%
STphosphatase	86	94.2%	96.6%	97.7%
TGF-beta	79	92.4%	98.7%	98.7%
TIM	40	92.5%	95.2%	100.0%
TNFR_c6	29	86.2%	92.3%	93.4%
UPAR_LY6	14	85.7%	77.8%	94.4%
Y_phosphatase	92	91.3%	93.4%	96.7%
Zn_clus	54	81.5%	84.5%	90.7%
actin	142	97.2%	96.9%	97.5%

Table 3: Results of Protein Classification using SMTs. The equivalence scores scores are shown for each model for the first 50 families in the database. The parameters to build the models were MAX_DEPTH= 7, MAX_PHI= 1 for the SMT prediction models and MAX_DEPTH= 5 and MAX_PHI= 1 for the SMT classifier model. Two tailed signed rank assigns a p-value to the null hypothesis that the means 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 $< 1\%$. Complete results are available at `http://www.cs.columbia.edu/compbio/smt/`.

Bailey, T. L., and Gribskov, M. 1998. Methods and statistics for combining motif match scores. *Journal of Computational Biology* 5:211–221.

Bairoch, A. 1995. The PROSITE database, its status in 1995. *Nucleic Acids Research* 24:189–196.

Baldi, P.; Chauvin, Y.; Hunkapiller, T.; and McClure, M. A. 1994. Hidden Markov models of biological primary sequence information. *Proceedings of the National Academy of Sciences of the United States of America* 91(3):1059–1063.

Bejerano, G., and Yona, G. 1999. Modeling protein families using probabilistic suffix trees. In *Proceedings of RECOMB99*, 15–24. ACM.

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. In Rawlings, C., ed., *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 47–55. AAAI Press.

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

Eddy, S. R. 1995. Multiple alignment using hidden Markov models. In Rawlings, C., ed., *Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology*, 114–120. AAAI Press.

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.

Gribskov, M.; Lüthy, R.; and Eisenberg, D. 1990. Profile analysis. *Methods in Enzymology* 183:146–159.

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. 1999a. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology.* To appear.

Jaakkola, T.; Diekhans, M.; and Haussler, D. 1999b. Using the Fisher kernel method to detect remote protein homologies. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, 149–158. Menlo Park, CA: AAAI Press.

Karplus, K.; Barrett, C.; and Hughey, R. 1998. Hidden markov models for detecting remote protein homologies. *Bioinformatics* 14(10):846–856.

Krogh, A.; Brown, M.; Mian, I.; Sjolander, K.; and Haussler, D. 1994. Hidden Markov models in computational biology: Applications to protein modeling. *Journal of Molecular Biology* 235:1501–1531.

Murzin, A. G.; Brenner, S. E.; Hubbard, T.; and Chothia, C. 1995. SCOP: A structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology* 247:536–540.

Pearson, W. R. 1995. Comparison of methods for searching protein sequence databases. *Protein Science* 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 variable 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 sequence homology. *Computer Applications in the Biosciences* 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.* Oxford UP. chapter Computer alignment of sequences, 59–72.

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