Using a Mixture of Probabilistic Decision Trees for Direct Prediction of Protein Function

Umar Syed and Golan Yona*
Department of Computer Science
Cornell University
* Corresponding author **email:golan@cs.cornell.edu**

ABSTRACT

We study the direct relationship between basic protein properties and their function. Our goal is to develop a new tool for functional prediction that can be used to complement and support other techniques based on sequence or structure information. In order to define this new measure of similarity between proteins we collected a set of 453 features and properties that characterize proteins and are believed to be correlated and related to structural and functional aspects of proteins. Among these properties are the composition and fraction of different groups of amino acids, predicted secondary structure content, molecular weight, average hydrophobicity, isoelectric point and others, as well as a set of properties that are extracted from database records of known protein sequences, such as subcellular location, tissue specificity, and others.

We introduce the mixture model of probabilistic decision trees to learn the set of potentially complex relationships between features and function. To study these correlations, trees are created and tested on the Pfam sequence-based classification of proteins and the EC classification of enzyme families. The model is very effective in learning highly diverged protein families or families that are not defined based on sequence. The resulting tree structure indicates the properties that are strongly correlated with structural and functional aspects of protein families, and can be used to suggest a concise definition of a protein family.

Keywords: sequence-function relationships, functional prediction, decision trees

Categories & Subject Descriptors: I.2.6 Learning, H.1.1 Systems and Information Theory.

General Terms: Algorithms, Theory.

1. INTRODUCTION

Given a new protein sequence, the first step toward predicting its function is usually through sequence analysis. The most basic form of sequence analysis is sequence comparison, in search of sequence similarity. This analysis is still the most common way for functional prediction today, and an increasing number of sequences in the protein databases are annotated using just sequence comparison.

Sequence similarity is only one way to quantify the similarity between proteins, but practically this is the only measure used today in database searches. However, in many cases sequences have diverged to the extent that their common ancestry cannot be detected even with the most powerful sequence comparison algorithms. When the structure is known, sequence analysis can be followed by structure comparison with the known structures in the PDB database. Since structure is more conserved than sequence, structural similarity can suggest functional similarity even when sequences diverged beyond detection. Yet, structural data is sparse, and is not available for newly sequenced genes.

When functional similarity is sought, other aspects of protein similarity should be considered beyond sequence and structure. Features such as the domain content, subcellular location, tissue specificity, organism classification and species, pairwise interactions, enzyme cofactors, catalytic activity, alternative products, and expression profiles may indicate common context and may suggest functional similarity even in the absence of clear sequence similarity (and sometimes, when structural data is known, even in the absence of structure similarity [1, 2]). These protein attributes and others are usually ignored, either because data is not available or because it is not clear how to use them to quantify similarity.

Here, we describe an effort to quantify functional similarity based on basic biochemical properties augmented with (partial) data available from database records on biological properties that may hint on the biological context of the protein. Incorporating new attributes and new similarity measures in the analysis of new genes can extend structure and function predictions to unknown domains of the protein space. This is extremely important in view of the sheer number of unidentified genes, and is useful in exploring high order organization within the protein space where even individual pieces of information can enhance greatly the clarity and accuracy of the suggested models.

Previous attempts to use alternative representations of proteins proved the potential of this approach. Some representations were based on dipeptide composition [3, 4] or combination of compositional properties and other physical/chemical properties [5]. In some cases these representations induced measures of similarity and dissimilarity between complete protein sequences that were used to classify the sequences into a fixed number of clusters [3], or to search

RECOMB'03, April 10–13, 2003, Berlin, Germany.

for close relatives [5]. In other cases, neural nets were applied to cluster the sequences based on the new representation [6, 4]. Other studies utilized information from multiple alignments to derive a new representation and study significant patterns in protein families [7, 8]. However, none of these methods was sensitive enough to become an effective tool for protein analysis and classification on a large scale. Perhaps the strongest limitation of these approaches was the lack of a model that can be tailored for each protein family. Rather, the general approach was to treat proteins as vectors in feature space. This approach limited the set of features that can be used to ordinal features, a subset of the physico-chemical properties considered here. Nominal features such as those derived from database records were never considered.

The focus of our study is to identify the most relevant and informative features and combination of features that best characterize protein families, and to create a model for each protein family that can be used for classification and function prediction. Our approach hinges on algorithms for decision tree building [9, 10, 11] but further expand the traditional work on decision trees, by introducing the **mixture model of probabilistic decision trees** (PDT). The trees handle missing attributes, ambiguities and fuzzyness that are expected when analyzing records of proteins by using a probabilistic framework, and are optimized to ensure high generalization power by testing several validation techniques.

The resulting tree structure indicates the properties that are strongly correlated with structural and functional aspects of proteins. This set of properties depends on the classification task. For example, the set of properties most strongly correlated with structural aspects are not necessarily the same as the properties that are most relevant for functional aspects. Clearly, this set can also change from one protein family to another. To study these correlations, trees were created and tested on two different types of known protein classifications: 1) sequence-based classification of proteins (i.e. Pfam [12]) 2) functional classification of proteins (i.e. the enzyme classification system [13]).

In the next sections we introduce our representation for proteins, the data extraction system, the decision tree model, and our new PDT model. We then turn to test it over the existing classifications.

2. THEORY AND METHODS

The learning system consists of two main parts: the first is the feature extraction system. Much thought was given to select an exhaustive set of attributes so as to create a reliable representation of a protein family. The second is the model used to identify regularities in the patterns of these attributes for each family - the set of attributes that best correlate with functional aspects of proteins and can most accurately predict membership in the family. The model that was developed for this task is a mixture of probabilistic decision trees. We first turn to describing the feature extraction system and then to the learning model.

2.1 Data preparation and feature extraction

Our basic data set is the SWISSPROT database release 39.3 and the TrEMBL database [14] release 14.4 as of July 15 2000, with 464744 proteins. A total of 453 features are used to describe each protein. The features are divided into three sets of features: features that can be calculated directly from the sequence, features that are predicted from the sequence and features that are extracted from database records.

2.1.1 Sequence features

These include composition percentages for the 20 individual amino acids, as well as for 16 amino acid groups adopted from [5] and [4], which are: charged (DEHIKLRV), positively charged (HKR), negatively charged (DE), polar (DEHKNQRSTWY), aliphatic (ILV), aromatic (FHWY), small (AGST), tiny (AG), bulky (FHRWY), hydrophobic (ILMV), hydrophobic aromatic (FWY), neutral and weakly hydrophobic (AGPST), hydrophilic acidic (EDNQ), hydrophilic basic (KRH), acidic (ED), and polar and uncharged (NQ).

Despite the overlap between these amino acid groups, each one has a unique characteristic. It is this characteristic that may play a role in defining and distinguishing family members from non-members, and therefore all of them were considered as distinct features.

We also calculated composition percentages for each of the 400 possible dipeptides. However, to save computation time, only the 20 most informative dipeptides were used during learning. The selection was done dynamically, during the training phase of the algorithm, on a per-family basis. We discuss this refinement further in section 2.2.3.

In addition to sequence composition, we also computed the average hydrophobicity, average isoelectric point, and average molecular weight of the protein sequence, as well as the overall sequence length. The hydrophobicities used are the scaled values, as described in [15]. The values for isoelectric point can be found at (http://www.ionsource.com/virtit /VirtualIT/aainfo.htm).

2.1.2 Predicted features

These represent the percentage of each protein that is a coil, helix, or a strand, as computed by PSIPRED [16], a secondary structure prediction program. For the sake of feasibility, we ran PSIPRED in 'single' mode for each sequence, meaning that the program did not search for sequence homologs to assist in its prediction.

2.1.3 Database features

We extracted nine database features from SWISSPROT [14]: three binary, five nominal, and one continuous (integer). The three binary features indicate the presence or absence of alternative products, enzyme cofactors, or catalytic activity in each protein, as determined from the optional fields of the CC section of each SWISSPROT protein record (see SWISSPROT user manual). A lack of annotation was interpreted as a '0' value; otherwise, the value was set to '1'.

The nominal features are more involved. Each protein is defined over zero or more values for each nominal feature: if $Values(A)$ is the set of all possible values for nominal attribute A, then each protein is defined over a subset of $Values(A)$. In all cases, a complete lack of information for A in a protein record was interpreted as an 'unknown' value for that attribute, and was treated specially by the decision tree algorithm (described in section 2.2.3). Two definitions of tissue specificity were included, one derived from the RC:TISSUE field, and one from the CC:TISSUE SPECIFICITY field. The subcellular location of the protein was derived from the CC:SUBCELLULAR LOCATION field,

while the organism classification and species were taken from the OC and OS fields, respectively.

Lastly, we also computed the number of PROSITE patterns exhibited by each protein by consulting the appropriate DR line of each record. Specifically, for each protein, we summed the number of certain and uncertain hits over all PROSITE patterns. This attribute is an indication of the domain/motif "complexity" of the protein.

It should be noted that we intentionally skipped the keywords in the KW field of SWISSPROT records. These keywords are strongly linked with protein function. However, as opposed to the other database attributes, these keywords explicitly describe protein functionality and are based on human annotation and interpretation. If ours was purely a classification task, the performance clearly could have improved by integrating SWISSPROT keywords. However, since we are interested in learning the mapping between proteins' properties and their function, we decided to exclude these keywords. We selected only database attributes that we have high confidence in and that are based on experimentally verified information.

2.2 The learning model

To model a protein family we developed a novel model, which we refer to as the **mixture model of probabilistic decision trees**. Our choice of the model was motivated by the nature of the data. Decision trees are useful when the data is nominal, i.e. when there is no natural notion of similarity or even ordering between objects. Such is the case for some of the attributes we associate with proteins, for example, tissue specificity and cellular location. These attributes rule out the use of many other popular machine learning models, such as Support Vector Machines. Other attractive aspects of decision trees are robustness to errors both in classification and attribute values. Moreover, decision trees can be used when some of the data is missing, and this is often the case with database attributes. We start by describing the traditional decision tree model, and then introduce our variations.

2.2.1 The basic decision tree model

Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each leaf node is terminal and is either associated with a category label or probabilities over different categories. Each internal node specifies a test of some attribute(s) of the instance and each branch descending from that node corresponds to a subset or range of the possible values of this attribute. An instance is classified by testing the attributes of the instance, one at a time, starting with the one defined by the root node, and moving down the tree along the branch corresponding to the specific value of the attribute in that instance, until you reach a leaf node. Note that this kind of scenario does not require a metric over instances.

2.2.2 The traditional training procedure for decision trees

When training decision trees from sample data S the goal is to create a concise model that would be consistent with the training data. Most learning algorithms use variations on a core algorithm that employs a top-down, greedy search through the space of all possible decision trees, and in this way organizes the tests in a tree. The algorithm will progressively split the set of training examples into smaller and purer subsets by introducing additional nodes with tests over new properties.

Usually, the criterion according to which the next attribute is selected is based on the reduction in impurity when testing on the attribute: the difference between the impurity before and after the split. One possible measure of the impurity or uncertainty is the entropy of the sample set at that node. The entropy impurity is also called the information impurity $i(S) = Entropy(S) \equiv -\sum_{j=1}^{c} p_j \log_2 p_j$ where p_j is the fraction of examples in the set S that are in category j. The common strategy is to select the attribute that decreases the impurity the most, where the drop in impurity is defined as:

$$\delta i(S) = i(S) - \sum_{v \in Values(A)} \frac{\mid S_v \mid}{\mid S \mid} i(S_v)$$

where $Values(A)$ is the set of all possible values for attribute A. When the measure used is the entropy impurity, the drop in impurity is called information gain and is denoted by $Gain(S, A)$

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{\mid S_v \mid}{\mid S \mid} Entropy(S_v)$$

The ID3 learning algorithm [17] uses this criterion to decide on which attribute to test next. The outline of the algorithm is given below

- Initialize a root node with the set of all samples S
- If no test reduces the node impurity
  - then node is a leaf
- else
  - select best test for decision node
  - partition instances by test outcome
  - construct one branch for each possible outcome
  - build subtrees recursively

Once the learning procedure terminated, the learned tree can be used to classify new instances. During classification, an example is repeatedly subjected to tests at decision nodes until it reaches a leaf, where it is assigned a probability equal to the percent of training examples at the leaf that are class members.

2.2.3 The extended training procedure for decision trees

The C4.5 algorithm [18] is an improvement over ID3 by using rule post-pruning to prevent overfitting, and several other modifications to account for bias in information gain introduced by multi-value attributes, and to handle missing attributes. Our basic procedure (a component of our new learning system) is similar to the C4.5 algorithm, with variations and enhancements that were adopted from different papers in this field, as well as several novel elements that we introduced into the learning procedure:

Dynamic attribute filtering: In the case of dipeptide composition, the number of attributes is too large to be used exhaustively during training. To narrow the scope of data that will be input to the decision tree algorithm, we need a filtering criterion that can examine the training data and retain only those attributes that seem important.

This importance is estimated by computing the ratio of the frequency among family members to the frequency among all non-family members. For example, for an attribute like 'AA', this ratio is the percent composition of 'AA' in family member sequences divided by the percent composition of 'AA' in non-family member sequences. Note that these calculations are restricted to the training set only. We retain only those attributes for which the ratio is very high or very low.

Discretizing numerical features: In a decision tree, numerical features must be divided into subranges before they can be used for splitting a node. Many researchers have approached this problem and the best solution is by no means clear. We use the Fayyad-Irani heuristic [19], an efficient and flexible method for discretizing numerical attributes. It is a greedy recursive protocol, inspired by the Minimum Description Length principle, that produces a partition of k-arity, where k is selected by the protocol itself. The protocol can also be constrained to produce only binary splits (see **Binary Splitting**, below).

Multiple values for attributes: The nominal database attributes that we use are somewhat different than those typically found in the literature on decision trees. Each example can take on several values for each nominal attribute, instead of just one. For example, a protein can be abundant in the liver as well as in heart tissues, and the number of possible combinations is huge. During training, when a node is split using a nominal attribute A, the set $Values(A)$ is partitioned across the node's children. With traditional nominal attributes, each example would be assigned to the unique child that corresponds to its value. In our model, however, each example may correspond to several children. To overcome that we divide the example itself across all children, weighting each child's portion by the number of values the example has in common with that child.

Missing attributes: Sometimes, an individual example will lack the attribute being tested by a decision node. We adopt the approach implemented in the C4.5 algorithm to handle these samples, and partition them across the child nodes maintaining the same proportions as the rest of the training examples (i.e. the proportions are calculated using only samples that have reached the splitting node and that have known values for the test attribute). These fractions can be partitioned again and again with every unknown attribute. Consequently, the ambiguous examples will reach several leaf nodes during classification. In these cases, the final probability is defined as the average of all leaf probabilities, weighted by the fraction of the example that reached each leaf.

Selection measures: The information gain measure is biased toward multi-valued attributes, when splitting these attributes over their many values. The C4.5 algorithm improves upon ID3 by using $GainRatio(S, A)$ [18]. However, $GainRatio(S, A)$ is itself less than ideal, since it may be biased toward attributes with very low $SplitInfo(S, A)$ instead high $Gain(S, A)$. To counter this, we also tested the closely-related $Mantras$ distance metric introduced by [20] who showed formally that his measure does not suffer from the limitations of $Gain(S, A)$ or $GainRatio(S, A)$ (though there are empirical evidences that it is not guaranteed to eliminate the bias altogether [21]). To simplify notation, in the following sections we refer to both $GainRatio$ and the $Mantras$ selection functions as **information gain**, unless specified otherwise.

Binary splitting: While many of our attributes are naturally suited to having multiple values, we wanted to test the impact of forcing all splits to be binary. This preserves training data for later splits, and consequently may improve accuracy, particularly since a very small fraction of our data set represents class (family) members. Moreover, binary splits are not susceptible to the biased entropy of many-way splits. With numerical attributes, the most straight-forward procedure is also the most efficient and exhaustive: simply restrict discretization to exactly 2 subranges (i.e. a single cut point). For a particular attribute, the optimal split can found in $O(N)$ time, where N is the number of possible cut points [19]. For nominal attributes, since there is no ordering over the values, the exhaustive procedure of testing every possible binary partition of the values leads to a time complexity of $O(2^d)$ where d is the number of possible values for the attribute $d = |Values(A)|$. However, we used the more sophisticated CART partitioning algorithm [22] which can find the optimal binary split in only $O(d)$, excluding sorting time.

Leaf weighting: Not all leaf nodes are good predictors and different nodes may have different accuracies, when tested on new samples. This is especially true if the number of samples in a node is small. To address that we tested also a variant where weights are associated with leaf nodes, and the final prediction is adjusted accordingly. The weights are trained using the perceptron learning algorithm [9], until a local minimum of the performance is achieved over the training set.

Post-pruning: We make intensive use of validation-based post-pruning in our algorithm. We refer to the set of samples available for learning as the **learning set** (the remaining samples are compiled into the **test set** used later for performance evaluation). The proteins in the learning set are divided into 10 equal parts. Let n be the size of the learning set. Ten different trees are generated for each family, and each one uses $\frac{9}{10}n$ as its **training set** and $\frac{1}{10}n$ as its **validation set**. Pruning is done in a bottom-up, node-by-node fashion. A node is pruned (i.e. converted to a leaf) as long as pruning it does not cause a decrease in performance over the validation set. The pruning decision is made locally, without considering the possible impact of future prunings higher up in the tree. This results in a pruning run-time of only $O(n)$, where n is the number of nodes in the unpruned tree. We evaluated performance in a variety of different ways, which are discussed in section 2.2.4. The final output is a performance-weighted average of the probabilities returned by each of the 10 trees. The cross-validation improves consistency and provides a more balanced model that learns the regularities that are common to different subsets of the data rather than those that are coincidental and happen only in a specific data set because of the random make up of samples.

2.2.4 *Introducing the mixture of probabilistic decision trees*

Our model addresses some of the fundamental problems with traditional decision tree learning algorithms. Specifically, we address four elements: optimization, evaluation, biased sample sets, and model selection. More precisely, we first propose an effective method of searching the hypothesis space that overcomes the pitfalls of the deterministic

learning algorithms. Secondly, we introduce a novel criterion function to evaluate decision tree performance. Thirdly, we describe a method of dealing with distributions in which negatives samples far outnumber positive samples, such as in our protein classification problem. Lastly, we propose an alternative method for deciding on the most probable model that is especially effective for small data sets.

The probabilistic framework: The basic procedure for building decision trees is deterministic. It is a greedy approach that will always select the attribute the maximizes the information gain at this point. However, this local maximization is not guaranteed to produce the "best" decision tree that describes the data. It happens quite often that several attributes have almost similar information gain values. The choice of the one that marginally outperforms others at that point may be proved to be less advantageous later on. Even if the chosen attribute has a significantly better information gain it still does not guarantee that in the optimal tree this attribute is indeed used at that point. To address this limitation imposed by the classical decision tree learning algorithm we switched to a probabilistic framework, in which the attribute is selected with probability that depends on its information gain

$$Prob(A) = \frac{Gain(S,A)}{\sum_i Gain(S,A_i)}$$

Thus, attributes with higher information gain have higher probability to be selected, but even attributes with small information gain have a non-zero probability to get selected. To diversify the composite model of a protein family, a total of 10 trees are learned for each training sets; one deterministic (using the traditional approach) and 9 probabilistic trees. With 10 training sets per family (see **Post-pruning** in section 2.2.3), the final mixture model has a total of 100 trees per family. The trees are weighted according to their performance over the learning set where the performance Q is measured in terms of the evaluation function (discussed in the next subsection). Denote by $P_i(+/x)$ the class probability of the sample x according to the i-th tree, then the total probability assigned by the hybrid mixture of trees is given by

$$P_T(+/x) = \frac{\sum_{i \in Trees} Q(i) P_i(+/x)}{\sum_{i \in Trees} Q(i)}$$

where $Q(i)$ is the quality (performance) of the i-th tree over the learning set (the higher, the better the tree separates the samples).

Evaluation of decision trees: A common measure of performance that is being used in many machine learning applications (including decision trees learning algorithms such as C4.5) is the accuracy, i.e. the percentage of correctly classified examples

$$accuracy = \frac{tp+tn}{tp+tn+fp+fn} = \frac{tp+tn}{total}$$

where tp is the number of true positives, tn number of true negatives, fp is the number of false positives, and fn is the number of false negatives. However, with the majority of the samples being negative examples, the accuracy may not be a good indicator of the discriminating power of the model. When the task is to discern only the members of a specific class (family) from a large collection of negative examples, better measures of performance are the sensitivity and selectivity (the positive predictive power), defined as

$$sensitivity = tp/(tp+fn) \qquad selectivity = tp/(tp+fp)$$

The categorization of samples as true(false) positives/negatives depends on the model and on the output it assigns to the samples. Since our decision tree outputs probabilities, we need a way to interpret these real numbers as either 'positive' or 'negative'. Usually one sets a threshold score T, a probability above which samples are predicted to be positive. Thus, if a sample reaches a leaf node j with membership probability $P_j(+)$ (defined based on the relative fraction of positive samples in this node), then it will be classified as positive if $P_j(+) > T$ (see Appendix A for discussion on methods for setting the threshold).

In addition to the accuracy and sensitivity/selectivity we introduce here a new evaluation function. This measure takes into account the complete distribution of positives and negatives and account for their statistical distance. Specifically we use the Jensen-Shannon (JS) divergence between probability distributions [23]. Given two (empirical) probability distributions $\mathbf{p}$ and $\mathbf{q}$, for every $0 \leq \lambda \leq 1$, the λ-JS **divergence** is defined as

$$D_\lambda^{JS}[\mathbf{p}||\mathbf{q}] = \lambda D^{KL}[\mathbf{p}||\mathbf{r}] + (1-\lambda) D^{KL}[\mathbf{q}||\mathbf{r}]$$

where $D^{KL}[\mathbf{p}||\mathbf{q}]$ is the relative entropy of $\mathbf{p}$ with respect to $\mathbf{q}$ (also called the Kullback-Leibler divergence [24]) and

$$\mathbf{r} = \lambda \mathbf{p} + (1-\lambda)\mathbf{q}$$

can be considered as the most likely common source distribution of both distributions $\mathbf{p}$ and $\mathbf{q}$, with λ as a prior weight (here we set $\lambda = 1/2$ since the weighted samples are divided equally between the two categories; see **Handling skewed distributions** below). We call the corresponding measure simply the **divergence score** and denote it by D^{JS}. This measure is symmetric and ranges between 0 and 1, where the divergence for identical distributions is 0.

All the measures provide rough estimates of the expected accuracy. However, unlike the other measures, the divergence score is less sensitive to outliers and is more likely to reflect the true estimate when applied to future examples. Moreover, there is no need to define a threshold T. Our goal is to maximize the statistical distance between the distribution of positives and the distribution of negatives, and create a separating buffer between the two distributions. The larger the separating buffer the better is the expected accuracy.

All three performance measures were tested during post-pruning (a node is pruned if the probabilities induced by the pruned tree are better at separating the positives from the negatives in terms of the evaluation function selected; see **Post-pruning** in section 2.2.3 for more details). Also, while testing a particular pruning technique, we always used that same evaluation function to compute $Q(i)$.

Handling skewed distributions: Since the majority of the sample set is composed of negative examples, one may run into problems when trying to learn small families. Because of its greedy nature most of the learning phase will concentrate on learning the regularities in the negative examples (that result in significant information gain). Not

only will this process cause extended learning times but it may also totally miss subtle regularities observed in the positive samples. The simple solution of presenting only a small set of negative samples equal in size to the set of positive samples is insufficient because some regularities observed in the positive set may be mistakenly considered as discriminating regularities, while if considered along with the whole set of negative examples they may be proved to be uninformative.

To overcome this problem we adopted a different approach where all the negative examples are presented, but are weighted equally to the total of positives examples (as suggested in [25]). Thus, every positive example contributes $1/\#positives$ and every negative example contributes $1/\#negatives$ to the total counts, and are weighted accordingly in the calculations of the information gain.

However, by doing so we introduce another problem. It may happen that we will end with a tree of high purity (when using the weighted samples), but of high impurity when considering the original unweighted samples, thus increasing significantly the rate of false positives. We tested two possible solutions that attempt to balance the two approaches. In the first, the criterion function for splitting and pruning is the average of the weighted-sample information gain with the unweighted-sample information gain (this protocol is called **mixed entropy**). The second solution starts by training the trees using the weighted samples. The usage of unweighted-sample entropy is delayed until the maximal information gain over all possible attributes at a given node is below a certain threshold. At that point each attribute is reconsidered using unweighted entropy and the training proceeds for as long as the information gain exceeds the threshold (we call this protocol **interlaced entropy**).

Note that the weighted samples are used only during the calculations of the information gain. For all other purposes, including pruning and prediction, the probabilities based on the unweighted samples are used.

Model selection by the MDL principle: One of the major problems with post-pruning methods is that a sufficiently large positive validation set is required to ensure high generalization power. Otherwise, in the presence of a very small positive learning set (and hence also a small validation set) many true regularities that are observed in the data set may be pruned since they are not supported by the small validation set. The effect of post-pruning in such cases can be drastic.

With very small data sets, where every single sample is important, one might want to consider alternative pruning methods that use all the given data. The chi-squared test is one possibility. However, it may suffer from what is called the *horizon effect* [9]. Here we use an alternative method based on the **minimum description length principle** [26] abbreviated as MDL.

The MDL principle is a common heuristic for selecting the most probable or valid model. It is based on the argument that was postulated by Occam in the 14th century. According to this argument (called *Occam's razor*), given two models that explain an observation equally likely, it is advised to select the simpler model, under the assumption that it will generalize better to new examples. The most probable model (of all possible models) for a given data set is the one that optimizes the complexity of the model and the quality of the explanation of the data. This is measured in terms of the description length. The description length of a given **model (hypothesis) and data** is defined as the description of **the model** plus the description of the **data given the model**.

One way to define the description length of a model is by its algorithmic complexity (Kolmogorov complexity). However, this approach is not very practical, since to define the algorithmic complexity of a model one has to find the shortest program that encodes for this model, something that is hardly ever possible.

A more practical approach, that formulates the same principle in different terms is the Bayesian approach. Formulated in probabilistic terms, the most probable model is the one that maximizes the posterior probability of the model given the data

$$P(h/D) = \frac{P(h)P(D/h)}{P(D)}$$

The optimal hypothesis h^* is the one that maximizes $P(h/D)$ or minimizes $-P(h/D)$

$$\begin{aligned} h^* &= \arg\min_h[-P(h)P(D/h)] \\ &= \arg\min_h[-\log_2 P(h) - \log_2 P(D/h)] \end{aligned}$$

By Shannon theorem $-\log_2 P(x)$ is related to the shortest description of a string x (in our case the model h), and the likelihood $-\log_2 P(D/h)$ is a measure for the description of the data given the model, therefore the MDL principle and the Bayesian approach as formulated above are practically the same. To find the optimal model we train a complete tree and then prune it in a bottom-up fashion, similarly to the post-pruning method described in section 2.2.3 using the new criterion function.

To approximate the first term we calculate the binary encoding of the model. For each node j we devote $\log N + \log k_j$ bits where N is the number of attributes in our model and k_j is the number of possible values for the attribute associated with node j. The total description length of the model is the sum of the descriptions of its nodes. Note that the number of possible models with the same description length increases exponentially, therefore the probability of the model decreases exponentially. Thus, the logarithm as applied to the probability of the model generates a term that is linear in the binary encoding of the model, as desired.

To calculate the second term we used two different approaches. The first calculates $P(D/h)$ based on the probabilities the model assigns to each sample. Specifically,

$$P(D/h) = \prod_{i=1}^{n} P_T(x_i) = \prod_{i=1}^{n} \sum_{j \in leaves(T)} f_j(x_i) Prob_j(x_i)$$

where $f_j(x)$ is the fraction of x_i that reaches leaf j and $Prob_j(x_i)$ is the probability assigned to x_i by that leaf. Intuitively, the likelihood term $\log P(D/h)$ accounts for the uncertainty that is left, given the completely determined model with all its parameters. With probability of 1 the uncertainty is zero, or in other words, the model describes the data perfectly. Otherwise, some uncertainty exists and we must include in the description the variance or the deviation of the data from the model. Only a combined description of the model and the variance will enable complete recovery of the data set.

The elimination of the validation set provides us with more training samples at our disposal. However, it also eliminates a stochastic element that can increase robustness, as it unifies all 10 training sets used by the post-pruning procedure into one. To compensate for the missing validation set in that aspect, we re-introduce 10 training sets by using the bootstrapping sampling method [27] of the learning set. Specifically, each training set of size n is generated by selecting instances from the learning set at random, with repetitions. Theoretical results [28] indicate that models estimated from bootstrap data sets will converge to the optimal model at the limit of infinite bootstrap data sets. Moreover, the bootstrap method was proved to be more robust in providing reliable estimates of performance in the presence of small data sets [29].

3. RESULTS

For training and testing we used two types of known protein classifications, the Pfam database of protein families and the EC database of enzyme families. There are many strategic decisions that one can take throughout the learning process. To find the optimal platform (learning algorithm) we first optimized the learning strategy over the Pfam data set (described in the next section). Note that this optimization does not use the test data to train the model, but merely to decide on the best protocol that should be employed. We then tested our model over both databases by training models for all families, using the optimal learning strategy, and testing them over unseen examples.

3.1 Optimization of learning algorithms

Having incorporated the modifications described in sections 2.2.3 and 2.2.4 into our model and the tree-generating program, we converged to a locally optimal learning strategy by conducting a rough greedy search over the **space of decision tree learning algorithms**. All performance evaluations are done over the off training set using the Pfam data set (see section 3.2) and averaged over all families. A step-by-step description of that search procedure follows:

The initial basic configuration is very similar to the C4.5 learning algorithm [18]. It uses multiple (variable) branching factor, unweighted entropy-based gain ratio, 5 cross-validation sets, accuracy-based post-pruning (with the confidence based threshold as described in Appendix A), unweighted leaf nodes in prediction and a smaller subset of 53 features after removing all dipeptide information. The performance of this configuration was poor (sensitivity of 0.35), and was gradually improved by considering each one of the modifications we discussed in section 2.2.4. The modifications were added one by one, and were accepted only if they improved performance over the previous configuration of the parameters. The results of this search are summarized in Table 1.

Switching to binary splitting, weighted entropy (see section 2.2.4) and Jensen-Shannon based post-pruning, the addition of dipeptides information and the introduction of the probabilistic framework improved performance and were accepted. Especially noticeable is the improvement due to the replacement of a single deterministic tree with the mixture of probabilistic trees. Increasing the number of cross-validation sets did improve the performance but only until a certain point (10 sets), beyond which no further improvement was observed. Leaf weighting and mixed entropy based information gain were rejected, as well as sensitivity/selectivity based pruning.

configuration	sensitivity (selectivity)	accepted/rejected
basic (C4.5)	0.35	initial
mantaras metric	0.36	accepted
binary branching	0.45	accepted
weighted entropy	0.56	accepted
10 cross-validation sets	0.65	accepted
20 cross-validation sets	0.64	rejected
JS-based post-pruning	0.7	accepted
sen/sel post-pruning	0.68	rejected
weighted leafs	0.68	rejected
mixed entropy	0.63	rejected
add dipeptides information	0.73	accepted
probabilistic trees	0.81	accepted

Table 1: Optimization of learning strategy. We start with a basic learning algorithm that is very similar to the C4.5 learning algorithm. Then, one step at a time, we introduce another modification to the learning algorithm and test its impact on the performance. Performance is measured in terms of the prediction sensitivity over the test set using the equivalence number criterion (see Appendix A). At the equivalence point, the sensitivity equals the selectivity.

The final configuration was set to the mixture of probabilistic decision trees including information on dipeptides (dynamically selected on a per-family basis, during training), with binary splitting, weighted entropy, 10 sets of cross-validation, and JS-based post-pruning and evaluation. The probability assigned by the mixture model to a sample x is defined as

$$P_T(+/x) = \frac{\sum_{i \in Trees} D^{JS}(i) P_i(+/x)}{\sum_{i \in Trees} D^{JS}(i)}$$

where $D^{JS}(i)$ is the divergence/separation score of the i-th tree over the validation set.

Due to time constraints, we were unable to exhaustively test the impact of the MDL-based pruning method on our algorithm. However, we did arrive at some preliminary results that suggest this modification is particularly effective at compensating for the deficiencies of validation-based post-pruning.

3.2 The Pfam classification test

Our first test is the Pfam database of protein domains and families. Of the 464744 proteins in the SWISSPROT and TrEMBL databases [14], 237925 proteins are classified into 2128 families (Pfam release 5.2). Most of the families in the Pfam database are domain families that characterize only a small part of the member proteins (these domains usually appear in conjunction with other domains). Since the properties we use in our model are global (calculated based on the complete sequence), only protein families that cover most of the sequence were considered for learning and evaluation. Specifically, we used a subset of 233 families with more than 80% coverage and at least 50 members. Because of the high coverage we expect that for these families the functionality is indeed associated with the complete protein chain.

Each family was divided into a learning set and a test set in the ratio of 3:1 so that 75% of the member proteins were used for learning, and 25% for testing, and a model was trained for each family using the optimal configuration as described in section 3.1. The mixture model was then used

to assign probabilities to each sample in the test set and the performance was evaluated by measuring sensitivity at the equivalence point (i.e. the point where the selectivity equals the sensitivity; see Appendix A for details).

In all cases, the mixture model performed better than the best of all individual trees. Interestingly, for almost half the families, the best probabilistic tree performed better (by up to 10%) than the greedy, deterministic tree, thus further supporting the strength of the probabilistic approach. In Fig. 1 we show the first five trees for the Pfam family Apolipoprotein. The first one is deterministic while the other trees were built using the probabilistic algorithm. The performance of the five trees is 0.96, 0.85, 0.84, 0.89, and 0.99, with the fifth probabilistic tree outperforming the deterministic tree.

To assess the power of our new model we compared it with BLAST [30], the most popular database search algorithm[1]. Since BLAST is a pairwise comparison algorithm we repeated the search for all query sequences in the family and we report the best performance as well as the performance of a typical BLAST search (averaged over all members). Table 2 gives the results for the 30 largest families. The average performance of the PDT model over all 233 families was 81%. A typical BLAST search detected 86% of the member proteins, while the best BLAST search detected 94% of the proteins on average.

Family name	Family size	Test set size	Percent testset detected by PDT	BLAST
GP120	21644	5411	1.00	- -
COX1	3397	850	0.97	1.00 (0.73)
MHC_II_beta	2348	587	0.97	1.00 (0.97)
F-protein	1637	410	0.99	0.99 (0.96)
Hemagglutinin	1360	340	0.99	1.00 (0.97)
p450	1328	332	0.78	0.97 (0.86)
globin	1064	266	0.89	0.87 (0.66)
adh_short	1012	253	0.63	0.97 (0.79)
rhv	895	224	0.95	0.84 (0.58)
vMSA	890	223	1.00	1.00 (0.91)
ras	793	199	0.86	0.99 (0.98)
NADHdh	786	197	0.94	0.99 (0.76)
Tat	707	177	0.99	1.00 (0.99)
adh_zinc	695	174	0.76	0.99 (0.84)
VPR	645	162	0.99	1.00 (0.98)
sugar_tr	624	156	0.65	0.91 (0.61)
Vif	604	151	0.99	1.00 (1.00)
fer4_NifH	561	141	0.85	0.96 (0.84)
actin	555	139	0.90	1.00 (0.97)
histone	536	134	0.95	0.54 (0.35)
cpn60_TCP1	516	129	0.85	0.95 (0.84)
HSP70	511	128	0.89	1.00 (0.99)
Gram-ve_porins	495	124	0.96	0.94 (0.82)
late_protein_L1	482	121	0.93	1.00 (0.76)
COX3	480	120	0.95	1.00 (0.94)
fusion_gly	442	111	0.99	1.00 (0.83)
HN	414	104	0.95	0.82 (0.51)
REV	375	94	0.95	0.97 (0.87)
serpin	353	89	0.78	0.95 (0.86)
COesterase	347	87	0.79	0.99 (0.92)

Table 2: PDT performance for the 30 largest Pfam families. Performance is evaluated using the equivalence number criterion (see text). For BLAST two numbers are given, for the best query and for a typical query (in parenthesis). The average performance of the PDT model over all 233 families was 0.81.

[1]A more appropriate test would be to compare our model with another statistical model. We are currently testing SAM [31], an Hidden Markov based search algorithm.

This test set was composed of mostly highly conserved protein families. With more than 80% sequence coverage it is clear that sequence similarity is very significant. Yet, even without explicitly using the information about the order of amino acids in the sequences, our model successfully captured the underlying principles for most protein families. For some families the PDT outperformed BLAST (e.g. rhv, histones, HN).

Our analysis so far suggests two main reasons why the model is short of detecting all member proteins for some of these homology-based protein families. The first is the use of weighted entropy. Although weighted entropy definitely improves performance (see table 1), it can also stop the learning too early, leaving some of the leaf nodes impure, and thus affecting performance. But perhaps a more significant flaw is the overly aggressive post-pruning. Post-pruning is based on a validation set, and for minor families, the validation set is especially small, and does not exhibit all the complex feature patterns of the protein family. In the search for the most concise representation of the family we may have eliminated some important branches that did not contribute to the performance over the validation set. We tried MDL-based pruning on a subset of 50 small families. The results were quite encouraging, boosting performance over this subset from an average of 0.34 to 0.47.

3.3 The information content of features

Overall, each protein sequence can be represented as a set of 453 attributes. Not all of them are equally important and not all of them are necessarily used in the model for each protein family. Those that are used are not exclusive in that they allow even proteins with unknown values for these attributes to be classified to the family. This is because of the nature of the learning system that can accommodate ambiguities by defining multiple rules for each protein family during the learning process.

To evaluate the quality of each feature as a protein family predictor, and to quantify its information content we trained 453 trees for each family, one for each feature. The tree was trained using the optimal configuration as described in the previous section. The results were averaged over all families and are given in Table 4. Results for just the dipeptide attributes are available in Table 5.

Clearly, no single attribute can serve as a good predictor in general. Performance vary between the different attributes and database attributes seem to have less information content than predicted and calculated attributes, perhaps because of the little information that is currently available in database records. For most proteins these attributes are undefined, thus affecting the discrimination power of the attribute.

Surprisingly, the organism attribute is the best predictor (information gain of 0.390). Note also its depth (5.6) meaning that it is usually being used earlier during the learning and decision process. Indeed many protein families exist only in a single kingdom or subphyla. The success of this attribute suggests that other database attributes can play a more significant role, as more data becomes available. The organism class is followed by the average hydrophobicity and the length. Other informative attributes are the relative compositions of different groups of amino acids, such as acidic, negatively charged, aromatic and so on.

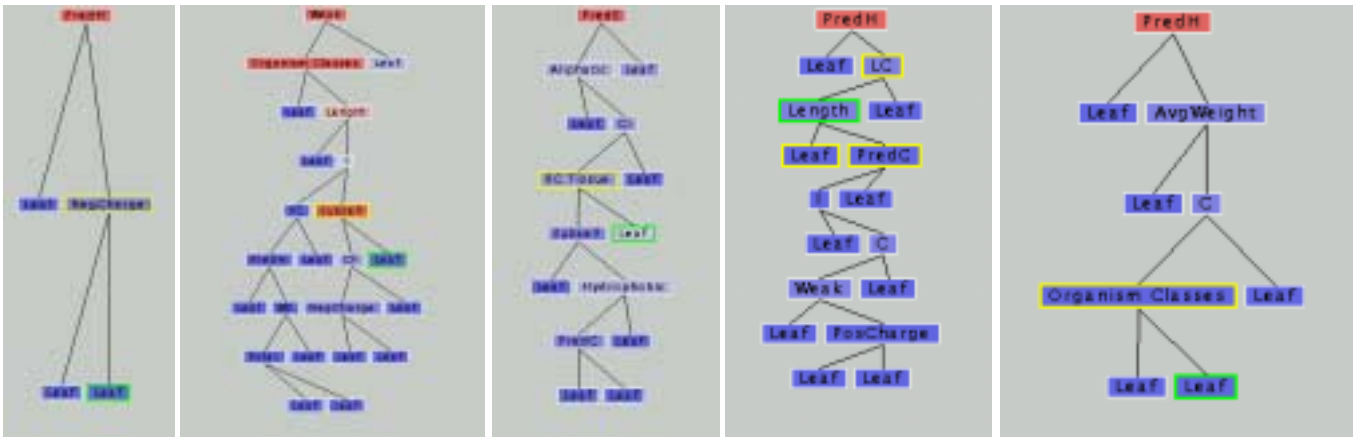

Figure 1: Alternative decision trees for Pfam family Apolipoprotein. The leftmost tree is a deterministic tree. The others are probabilistic. The rightmost tree performed the best.

3.4 The EC classification test

Since Pfam is not necessarily a functional classification but rather a domain based classification, many of the features that we associate with complete protein chains may not be correlated with the specific domain. It is possible to "localize" some features and calculate them just along the domain region. However, some features, such as database attributes, cannot be localized.

While the Pfam test establishes the validity and proves the potential of our method, it is the EC classification test that shows the true prediction power in cases where homology based methods fail. It is well known that proteins need not to be homologs to perform similar functions. Many families of proteins that perform similar functions are defined based on their functional role in the cell rather than based on their common evolutionary ancestry. In some cases no such kinship can be established, and there are examples in the literature of protein that share no sequence or structure similarity and yet perform similar functions [1, 2]. However, since detecting this kind of similarity is much more difficult and is involved with determining the functionality of the proteins in vivo not much information is available about these protein families. Probably the most extensive collection of functionally-based protein families is the EC classification system. To test our model on this functional classification we extracted all enzyme families and trained models for all families with more than 50 members, for a total 229 families. The results are listed in table 3 for the 30 largest families. The average performance of the PDT model over all 229 families was 71%. Note that in this case the PDT model outperformed the best BLAST in many cases, especially when the enzyme family is composed of several sequence-based families.

4. DISCUSSION

The gene data that is available today presents us with a major challenge. How to decipher the rules that define protein functionality. Obviously this has a bearing to the sequences of these genes. According to the central dogma of molecular biology, it is the protein sequence that dictates the structure of the protein and this structure in turn prescribes the biological function of the protein. Moreover, sequence similarity can suggest common heritage and similar functions. Traditionally, sequence analysis has been playing a significant role in predicting gene function. However, there is a growing number of protein families that cannot be defined based on sequence similarity, and are linked and grouped based on other features. Some may argue that the tools that are available today for sequence analysis already exhaust sequence information. Indeed, in some cases it seems that new evidence must be presented, either in the form of structural similarity, expression data or protein interaction data before relationships can be established.

Here we argue the opposite. Our ability to analyze an entity and explore its relationships with other entities depends on the representation we employ for that entity. Traditionally proteins were analyzed as sequences of amino acids. This fairly simple representation was complex enough to detect many relationships. But once the order of amino acids was eliminated much of the information was lost. In this paper we present a new representation for proteins, through an exhaustive set of attributes, and introduce a novel model to decipher the regularities that distinguish family members from non-members. The model uses only sequence data, yet it can detect subtle principles that not always can be established based on pure sequence similarity.

The set of attributes covers a broad range of protein features, from basic compositional properties to physico-chemical features that characterize and are related to the cellular environment in which the protein functions, predicted features that hint at the topology and the shape of the protein structure, and database attributes that provide additional information about the subcellular location of the protein, its tissue preferences and other feature that are likely to be linked to protein functionality. This set of features can be extended as more data becomes available. The combination of nominal attributes with ordinal attributes in our model suggested the use of decision trees, one of the very few machine learning techniques that can handle mixed data. In addition to being suited for our learning task, the model can tolerate noise and missing data, as is often the case with database attributes. However, even the most recent learning algorithms for decision trees proved unsuccessful and the accuracy of the learned models was insufficient for effective prediction. Therefore we embarked on developing a new model that we

Family name	Family size	Test set size	Number subfamilies	Percent testset detected by PDT	BLAST
4.1.1.39	3879	970	2	0.99	0.95 (0.91)
1.9.3.1	1904	476	18	0.95	0.85 (0.47)
3.6.1.34	1612	403	26	0.87	0.42 (0.23)
2.7.7.48	488	122	21	0.95	0.37 (0.14)
2.7.7.6	473	119	26	0.64	0.42 (0.18)
2.7.7.7	461	116	13	0.67	0.45 (0.24)
3.1.3.48	395	99	3	0.85	0.83 (0.59)
1.14.14.1	382	96	1	0.76	0.70 (0.58)
2.7.1.112	367	92	2	0.79	0.92 (0.82)
1.1.1.1	360	90	2	0.91	0.77 (0.61)
5.99.1.3	320	80	2	0.81	0.92 (0.66)
1.2.1.12	318	80	1	0.88	0.98 (0.94)
1.6.99.3	315	79	37	0.68	0.40 (0.18)
2.7.7.49	284	71	1	0.77	0.80 (0.57)
5.2.1.8	245	62	2	0.76	0.72 (0.55)
2.5.1.18	240	60	5	0.77	0.77 (0.49)
3.2.1.14	222	56	2	0.73	0.51 (0.31)
6.3.1.2	213	54	1	0.87	0.97 (0.86)
3.1.3.16	211	53	9	0.77	0.67 (0.46)
3.2.1.4	207	52	2	0.40	0.43 (0.23)
1.11.1.7	179	45	5	0.76	0.82 (0.66)
3.1.1.4	175	44	3	0.93	0.91 (0.82)
1.11.1.6	170	43	3	0.91	0.92 (0.82)
3.2.1.1	168	42	2	0.79	0.76 (0.57)
1.18.6.1	168	42	2	0.74	0.57 (0.42)
3.4.99.46	161	41	1	0.88	0.98 (0.73)
2.7.2.3	157	40	2	0.80	0.97 (0.93)
3.1.26.4	154	39	2	0.69	0.50 (0.33)
4.1.1.31	150	38	2	0.92	0.99 (0.85)
3.5.2.6	150	38	3	0.74	0.73 (0.52)

Table 3: PDT performance for the 30 largest EC families. All columns but the forth are the same as in table 2. Fourth column is the number of different sequence subfamilies (Chau and Yona, unpublished work). The average performance of the PDT model over all 229 families was 0.71.

call probabilistic decision trees based on the original decision tree model. In addition, we introduce several other components and options in the learning algorithm, such as different pruning methods and different weighting schemes. To find the optimal learning strategy we search through the space of decision tree learning algorithms until we converge to a locally optimal strategy. Four main elements characterize our final model; (1) dynamic feature and value selection for large feature and value sets, (2) probabilistic attribute selection protocol, (3) the use of mixture of trees, and (4) the use of positive and negative sample divergence properties in all stages of tree learning, from pruning to tree weighting and evaluation. We use the Jensen-Shannon divergence measure to assess the trees in terms of the separation they induce over the data set between the positive and negative samples (members and non-members). Trees that maximize the separation are assigned a higher score (weight) and play a more dominant role in prediction. Thus the learning process essentially maximizes the buffer between the true and negative samples, without explicitly optimizing a specific score function. One might notice the similarity with kernel-based methods such as the Support Vector Machines that try to maximize the margins around the separating hyper-planes.

Our model performed significantly better than the classical tree learning algorithms and the performance leaped from 0.35 success rate to 0.81. The fundamental principle that explains the success of the mixture model is the same underlying principle behind other machine learning techniques that proved to be more powerful than others for certain problems. Namely, the Bayesian parameter estimation approach, the Bayes optimal classifier, the Gaussian mixture model and Boosting. The common thread among these different models is the integration (summation) over many possible component hypotheses. While the original hypothesis space selects a particular representation (that can be based on our prior set of beliefs), thus imposing a structure and constraining the expressive power of the model, the integration (summation) over the component models enables us to model complex target functions that cannot be modeled effectively by the underlying component model. By combining the predictions of all possible hypotheses we overcome the restriction we imposed with the original representation, and we are no longer confined to this representation. Note that this is equivalent to estimating the likelihood of a new instance x directly from the training data D, while integrating over all possible models, i.e.

$$p(x/D) = \int p(x/\theta)p(\theta/D)d\theta$$

where θ is the model parameter vector (in the case of a decision tree, θ represents the set of tests in a specific tree, and the integration is being replaced with summation).

To assess our model with respect to the available knowledge on protein families and existing models we evaluated it over two well known classifications; the Pfam homology based classification and the EC functional based enzyme classification. The results were compared to the popular BLAST algorithm. Indeed our model compares favorably with BLAST over the Pfam data set, but the power of our model is more pronounced when the protein family is defined based on function as in the EC database, rather than based on homology. One of the advantages of our method is that the sequences need not to be aligned. The model can learn the features common to a diverse set of proteins of shared function, sometimes without even evident sequence homology. When these features are clearly a property of the protein family and are not found in other sequences then they serve as good predictors and are integrated into the model.

There are several modifications that may improve performance. One modification that we are considering is to modify the pruning algorithm (using global as opposed to local optimization). Another modification would be to assign probabilities to attribute values since for some nominal attributes it is possible to quantify the likelihood of the different values. Clearly, integration of other features can refine the models, and finally, boosting techniques can also help to improve the performance and we intend to test them in the future.

5. ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under Grant No. 0133311 to Golan Yona.

6. REFERENCES

[1] Wilson, D. B. & Irwin, D. C. (1999). Genetics and properties of cellulases. *Adv. Biochem. Eng.* **65**, 2-21.

[2] Stawiski, E. W. , Baucom, A. E. , Lohr, S. C. & Gregoret, L. M. (2000). Predicting protein function from structure: unique structural features of proteases. *Proc. Natl. Acad. Sci. USA* **97**, 3954-3958.

[3] van Heel, M. (1991). A new family of powerful multivariate statistical sequence analysis techniques. *J. Mol. Biol.* **220**, 877-887.

[4] Ferran, E. A., Pflugfelder, B. & Ferrara P. (1994). Self-Organized Neural Maps of Human Protein Sequences. *Protein Sci.* **3**, 507-521.

[5] Hobohm, U. & Sander, C. (1995). A sequence property approach to searching protein database. *J. Mol. Biol.* **251**, 390-399.

[6] Wu, C., Whitson, G., Mclarty, J., Ermongkonchai A. & Chang, T. (1992). Protein classification artificial neural system. *Protein Sci.* **1**, 667-677.

[7] Han, K. F. & Baker, D. (1995). Recurring local sequence motifs in proteins. *J. Mol. Biol.* **251**, 176-187.

[8] Casari, G., Sander, C. & Valencia, A. (1995). A method to predict functional residues in proteins. *Nat. Struct. Biol.* **2**, 171-178.

[9] Duda, R. O., Hart, P. E. & Stork, D. G. (2000). "Pattern classification". John Wiley and Sons, New York.

[10] Mitchell, T. M. (1997). "Machine Learning". McGraw-Hill.

[11] Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1993). "Classification and Regression Trees". Chapman & Hall, New York.

[12] Bateman, A., Birney, E., Durbin, R., Eddy, S. R., Finn R. D., & Sonnhammer E. L. (1999). Pfam 3.1: 1313 multiple alignments and profile HMMs match the majority of proteins. *Nucl. Acids Res.* **27**, 260-262.

[13] http://www.chem.qmw.ac.uk/iubmb/enzyme/

[14] Bairoch, A. & Apweiler, R. (1999). The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1999. *Nucl. Acids Res.* **27** 49-54.

[15] Black, S.D. & Mould, D.R. (1991). Development of Hydrophobicity Parameters to Analyze Proteins Which Bear Post or Cotranslational Modifications. *Anal. Biochem.* **193**, 72-82.

[16] McGuffin, L. J. , Bryson, K. & Jones, D. T. (2000). The PSIPRED protein structure prediction server. *Bioinformatics* **16**, 404-405.

[17] Quinlan, J.R., (1986). Induction of decision trees. *Machine Learning*. **1**, 81-106.

[18] Quinlan, J.R., (1993). "C4.5: Programs for Machine Learning". Morgan Kaufmann.

[19] Fayyad, U. M. & Irani, K. B. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proc. of the 13th int. conf. on AI*, 1022-1027. Morgan Kaufmann, San Mateo, California.

[20] Mantaras, R. L. (1991). A Distance-based Attribute Selection Measure for Decision Tree Induction. *Machine Learning* **6**, 81-92.

[21] Kononenko, I. (1995). On biases in estimating multi-valued attributes. In *Int. Conf. on AI.* 1034-1040.

[22] Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1993). "Classification and Regression Trees". Wadsworth Int. Group, Belmont, California,

[23] Lin, J. (1991). Divergence measures based on the Shannon entropy. *IEEE Trans. Info. Theory* **37:1**, 145-151.

[24] Kullback, S. (1959). "Information theory and statistics". John Wiley and Sons, New York.

[25] Eskin, E., Grundy, W. N. & Singer, Y. (2000). Protein Family Classification using Sparse Markov Transducers. *In the proceedings of ISMB 2000*, 20-23.

[26] Rissanen, J. (1989). Stochastic Complexity in Statistical Inquiry. *World Scientific.*

[27] Hjorth, J. S. U. (1994). "Computer intensive statistical methods validation, model selection, and bootstrap". Chapman & Hall, London.

[28] Jain, A. K. , Dubes, R. C. & Chen, C. (1998). Bootstrap techniques for error estimation. *IEEE Transactions on Pattern Analysis and Applications* **9**, 628-633.

[29] Shakhnarovich, G. , El-Yaniv, R. & Baram, Y. (2001). Smoothed bootstrap and statistical data cloning for classifier evaluation. *ICML-2001*

[30] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W. & Lipman, D.J. (1997). Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucl. Acids Res.* **25**, 3389-3402.

[31] Hughey, R. & Krogh, A. (1998). "SAM : Sequence alignment and modeling software system". Technical Report UCSC-CRL-96-22, University of California, Santa Cruz, CA, July 1998.

[32] Pearson, W. R. (1995). Comparison of methods for searching protein sequence databases. *Protein Sci.* **4**, 1145-1160.

7. APPENDIX A - SETTING THE THRESHOLD FOR EVALUATION AND PREDICTION

When evaluating the performance of a model that assigns probabilities to samples one needs a clear definition of positives and negatives. Usually a threshold T is defined, and if the sample probability exceeds this threshold, then the sample is defined as positive.

How should one define the threshold T? We tested two different approaches. The naive approach would be to take a majority vote, so that all samples with probability higher than 0.5 are defined as positive. However, this may be misleading, since the number of positives and negatives may differ significantly to begin with. A more sensible approach would be to take into account the prior probabilities of positives $P_0(+)$ and negatives $P_0(-)$, and set the threshold to $P_0(-)$. To account for random fluctuations due the varying sample sizes at the nodes, a different significance threshold T_j is calculated for each node. Specifically, if the total number of samples in a leaf node j is N_j then the number of positives that will reach this node by chance (the null hypothesis) is expected to follow a binomial distribution with mean $N_j \times P_0(+)$ and variance $N_j \times P_0(+) \times P_0(-)$. Using the normal approximation for the binomial distribution, we set the threshold at the 90% precentile of the distribution, and if the node probability $P_j(+)$ exceeds this threshold then we define this instance to be positive. Thus, each example is classified unambiguously in a discrete manner, and the accuracy is computed thereby. If the sample reaches a single leaf node then only the confidence interval for this node is being used to assign a label. If it reaches more than one node, a "combined" node is created where the datasets are collected from the different leaf nodes, weighted by the fractions of the sample that reach each leaf node, and the sample probabilities, the threshold and the label are calculated accordingly. We refer to this approach as the **confidence-based** approach.

The second approach uses the equivalence number criterion. All sequences in the database are sorted according to the output assigned by the model. The equivalence point is the point where the number of false positives equals the number of false negatives [32]. All proteins that are predicted with higher probability are considered as positives. In other words the equivalence point is the point that balances sensitivity and selectivity. We refer to this approach as the **equivalence-point** approach.

Attribute	Information gain	Depth
OrganismClasses	0.390	5.6
AvgHydro	0.289	6.4
Length	0.259	6.3
NegCharge	0.253	6.5
Acid	0.253	6.7
Aromatic	0.245	6.9
HydrophobicArom	0.238	7.0
HydrophilicAcid	0.225	7.0
AvgWeight	0.223	6.9
Bulky	0.210	7.2
PosCharge	0.209	6.9
HydrophilicBase	0.209	6.9
E	0.209	7.2
Charge	0.207	6.8
Hydrophobic	0.202	6.9
PredH	0.200	6.7
AvgIso	0.196	7.2
Polar	0.191	6.7
W	0.190	7.4
R	0.190	7.4
Aliphatic	0.187	6.8
PrositeDomains	0.185	7.5
PredC	0.184	6.9
D	0.182	7.4
Y	0.179	7.6
Tiny	0.176	7.0
L	0.172	7.4
G	0.168	7.3
K	0.166	7.5
C	0.163	7.4
PredE	0.162	7.0
Weak	0.161	7.2
Small	0.160	7.3
F	0.155	7.5
Proline	0.147	7.4
P	0.147	7.4
A	0.142	7.3
V	0.141	7.6
T	0.135	7.7
PolarUn	0.127	7.5
N	0.121	7.7
H	0.121	7.8
I	0.110	7.7
S	0.099	7.9
Species	0.098	6.7
Q	0.098	7.9
M	0.098	7.9
Catalytic	0.089	8.4
Cofactor	0.039	9.7
Subcell	0.035	6.8
RCTissue	0.011	7.5
Alternate	0.006	10.7
TissueSpec	0.002	6.9

Table 4: Attribute information - part 1 (excludes dipeptides). Attributes are sorted by decreasing information gain with respect to that attribute only (second column). The results are averaged over all families for which the attribute is defined. Since throughout the learning phase we use the weighted entropy (see section 2.2.4) the information gain in this case is given over the weighted samples. These numbers are also more effective for identifying informative attributes; because of the skewed distribution of positives and negatives the background unweighted entropy is very small to begin with and significant improvement in prediction power will translate only to slight improvements in the unweighted entropy (the non-linearity of the entropy function further obscures significant changes in classification accuracy). Third column is the average tree depth where the attribute is used for the first time (the depth of the root node is 0). The average depth of a tree is 16.83.

Attribute	Information gain	Depth
DP	0.192	7.9
GG	0.180	6.6
YG	0.171	4.8
EL	0.167	5.5
LD	0.166	—
GA	0.166	5.4
DG	0.164	8.4
GL	0.163	6.6
PL	0.162	—
IG	0.161	5.9
GR	0.160	6.9
LY	0.157	—
AG	0.157	6.9
LG	0.156	4.0
FL	0.155	7.1
FF	0.154	8.0
RR	0.149	6.6
EV	0.149	—
EG	0.149	9.2
EE	0.149	6.6
WL	0.148	7.7
QL	0.148	5.9
DV	0.148	—
VD	0.147	—
NG	0.147	3.8
LE	0.147	6.1
KK	0.147	7.0
EA	0.147	7.0
AV	0.147	—
NV	0.146	5.4
GK	0.146	7.9
DL	0.145	—
VR	0.144	5.2
IL	0.144	—
AI	0.144	7.1
AA	0.144	6.4
MP	0.143	7.1
GT	0.143	5.6
NP	0.142	8.6
LR	0.142	6.6

Table 5: Attribute information - part 2 (dipeptides only). See Table 4 for details. Missing depth information means that the dipeptide was never used under the optimal configuration of the model. Only the top 40 dipeptide attributes are listed.