# Machine Learning in Bioinformatics: Algorithms, Implementations and Applications

Robert Ezra Langlois

B.S. (University of Illinois at Chicago) 2002
Doctor of Philosophy in Bioinformatics

# Contents

# Summary

Machine learning has proven to be a crucial methodology to solving problems in many domains ranging from finance to marketing to biology. The biological applications have shown great promise in replacing some "web-lab" experiments, guiding experiments and elucidating the underlying mechanisms governing the corresponding biological processes. In board terms, the goal of this work is to develop a machine learning framework from which to study the structure, function, evolution and mutation of proteins in order to gain a better understanding of the corresponding biological processes they comprise. First, an open-source machine learning workbench was developed to serve as a platform for this work. Further, I have developed a new algorithm that modifies the AdaBoost algorithm to handle the multiple-instance learning problem. Second, I have applied this workbench to the problem of understanding the effect or disease-association of a mutated amino acid in the protein. This problem is broken down into three steps. The first step uses a multi-classification algorithm to assign a sequence to one of a set of folds (three-dimensional structures). Using this structure, the second step assigns the function of the protein, which places any deleterious mutation in the context of the cell. Also from the structure, the third step predicts whether the given mutation affects the function of the protein. Third, I have applied this workbench to the identification of DNA-binding proteins and their corresponding functional sites as an initial step to building a larger transcription factor regulatory network. Indeed, in this work, I have developed a new sequence-based feature representation that implicitly encodes potential important residues. Fourth, I have extended this work on DNA-binding protein identification to the genome-scale where I analyze the evolutionary relationships of DNA-binding proteins between organisms. Interestingly, I found a discrepancy between phylogenetic trees grown using genome sequence similarity and classification similarity using this new feature representation where the classification similarity proposes a more reasonable phylogenetic tree making Saccharomyces cerevisiae (a yeast) the ancestor of Arabidopsis thaliana (a plant).

# Chapter 1

# Introduction

I present a general machine learning methodology and a practical algorithm to study biological problems. Indeed, both biology and medicine provide a myriad of interesting problems for machine learning having large amounts of data stockpiled in massive databases. In this work, I focus on problems related to protein function and structure with an emphasis on DNA–binding proteins. While each problem is important in its own right, protein–DNA interactions serve a fundamental role in cellular regulation and function. Moreover, the importance of a number of these less-studied interactions (*e.g.* chromatin remodeling) is recognized in the emerging field of epigenetics Jaenisch and Bird (2003).

## 1.1 *In Silico* Experiments

Even before high throughput techniques, biologists have collected large amounts of data that ends up in large repositories. Using simple analytical techniques, a biologist could derive important correlations and subsequently conclusions from this data. Indeed, many experiments often serve to guide a future, more important or expensive experiment. Likewise, unable to see the underlying mechanisms, many models have been proposed based on such data and in too many cases debunked by future data. Moreover, standard sequence and structural analysis techniques fail in many cases to accurately pick up experimental slack.

### 1.1.1 *In lieu* of Wet Experiments

One of the long term goals of this work is to provide a cheaper computational alternative to wet lab experiments. It is not uncommon for an experiment to be repeated on many different targets. For example, crystallography experiments have been repeated for upwards of several thousand distinct proteins. Likewise, footprinting assays have been

used to identify hundreds of proteins that bind DNA. This vast accumulation of data may hold enough information to characterize a target (protein) rendering the original "wet" experiment obsolete. These *in vivo* (*in vitro*) experiments will be replaced by *in silico* experiments that rely on previously determined targets to predict the experimental outcome of a new target. For example, given all the previously established protein–DNA interactions it will become possible to determine if an unseen protein binds DNA based on the protein's sequence and/or structural characteristics.

### 1.1.2 Underlying Mechanisms

Another goal of this work is to understand the underlying mechanisms that govern the outcome of a particular experiment. In other words, a machine learning algorithm constructs a model in order to predict the outcome of some experiment. If the model generalizes the data well (is accurate) then an analysis of the model will give some insight into the underlying mechanisms that govern the targets reaction to a particular experiment. For example, say one trains a classifier over a set of examples that contains proteins that both bind DNA and do not. The model generated by the classifier holds important characteristics that define DNA–binding and could lead to a better understanding of interactions necessary to facilitate DNA–binding.

### 1.1.3 A Guide to Experimentalists

The final goal of this work is to guide experimentalists in selecting experiments or conditions of an experiment. So far, this is the most realistic goal of the current work. While we cannot definitively identify the outcome of an experiment on a target, we can select a subset of interesting targets from a larger group or suggest a particular experiment for a target of interest. For example, our DNA–binding protein prediction results in conjunction with CHIP–chip would suggest further experiments on proteins where there is a strong disagreement in the results.

### 1.1.4 Alternative to Sequence/Structure Analysis

A well known computational technique that is widely applicable to many biological problems is sequence and (to some degree structural) analysis. These database searching techniques use homology or conserved motifs to assign function to a target protein. However, there are many cases when sequence similarity does not transfer function. For instance, despite being highly homologous, the affinities of the FYVE lipid-binding domains differ substantially Blatner *et al.* (2004). A more reliable prediction can be made using the three-dimensional fold of the protein where about two-thirds of the time, similar folds have the same function Koppensteiner *et al.* (2000). However, the left over one-third have many striking examples where proteins with a similar fold have widely different functions.

Two examples pertain to the helix-turn-helix DNA–binding motif and the PH domains for the lipid binding domains. Indeed, many helix-turn-helix proteins bind DNA and it is considered an archetypical motif. However, a good portion of proteins with this fold do not have any interaction with DNA Shanahan *et al.* (2004). Similarly, the PH domain is often associated with lipid-binding; however, there are a number of protein examples with the PH domain where it serves another function Lemmon and Ferguson (2000); Singh and Murray (2003).

## 1.2 Select Biological Problems

This section addresses three important, distinct biological problems that can be cast in the classification setting. Firstly, we introduce protein–membrane interactions, an understudied field in proteomics. Then, we introduce the problem of recognizing disease causing single amino acid polymorphisms (or SAPs). Finally, we introduce the difficult problem of assigning a protein sequence to a particular three-dimension fold.

While these problems are distinct, together they from a framework for understanding genetic disease association. That is, often a researcher only has a sequence and a proposed mutation. The first step is to model the structure of the region with this mutation (Fold Recognition). The second is to then annotate the protein (here we focus on protein–membrane and protein–DNA interactions) based on either or both sequence and structure in order to find potential functional sites. The third and final step is to predict whether the mutation may cause a disease (Disease Causing SAPs) in the protein. This framework gives both an accurate prediction of disease and the possible pathway or mechanism this diseased mutation disrupts.

### 1.2.1 Protein–fold Recognition

While a similar structure is sometimes not sufficient to transfer function from a known case to a target, it still provides a rich set of features to identify function. When structure is unknown, a number of experimental and computational techniques are available to elucidate structure from sequence. One one hand, Experimental techniques include both x–ray crystallography Crowfoot (1935) and nuclear magnetic resonance Wuthrich (1990) and solving a structure using these techniques takes months to years. On another hand, computational methods depend on the available data; when a target is similar in terms of sequence to a protein with a solved structure then it often shares a very similar structure. When no such similar sequence exists, techniques such as threading Sanchez and Sali (1997); McGuffin and Jones (2003); Xu and Xu (2000); Skolnick and Kihara (2001) and de novo folding Bonneau *et al.* (2001); Kihara *et al.* (2001) attempt to approximate the fold. However, these techniques exploit only the simplest relationships between features and perform at best moderately well. Thus, this work presents a principled method to

classify a protein sequence into one of many template structures using machine learning.

### 1.2.2 Disease Causing Polymorphisms

With the completion of the human genome project, attention has shifted to human genomic variation. One type of variation that has received commiserate interest of late is the single nucleotide polymorphism (SNP). With an average density of 1 in 300 base pairs, SNPs account for a good deal of the individuality and diversity in the human population Consortium (2003); Kruglyak and Nickerson (2001); Reich *et al.* (2003); Pastinen *et al.* (2006). A SNP that causes an amino acid substitution in the protein product is known as a non-synonymous SNP (*ns*SNP) or also as a single amino acid polymorphism (SAP)). Indeed, SAPs account for about 50% of the genetic diseases caused by SNPs Krawczak *et al.* (2000). To this end, large scale efforts such as the HapMap project Consortium (2003) have accumulated considerable SAP related data in databases like dbSNP Sherry *et al.* (2001). However, these high–throughput experiments fail to experimentally characterize SAPs in terms of disease association. Furthermore, the underlying mechanisms that lead a SAP to cause a disease is poorly understood. Thus, this work introduces a machine learning protocol to identify SAPs that lead to disease and subsequently analyzes the importance of individual features in identifying disease causing SAPs.

### 1.2.3 Protein–membrane Interactions

Protein–membrane interactions serve a number of cellular processes including cellular signalling. Interest has grown considerably in recent years in one such group of proteins, known as peripheral membrane-binding proteins, which, localize to the membrane in order to find their binding partners Cho (2001); Hurley and Meyer (2001); Teruel and Meyer (2000). These membrane-binding proteins reversibly bind lipids in the membrane using a number of mechanisms including specialized domains or even just a specialized surface area. Current experimental techniques to study peripheral membrane–binding proteins include surface plasmon resonance (SPR) analysis Mozsolits and Aguilar (2002) and fluorescence resonance energy transfer (FRET) analysis Wu and Brand (1994). These techniques are prohibitively expensive for any large-scale analysis.

## 1.3 Protein–DNA Interactions

DNA–binding proteins maintain, regulate, read and replicate the fundamental code of life, DNA. These proteins comprise roughly 7% of proteins encoded in the eukaryotic genome and 6% in the prokaryotic genome Luscombe *et al.* (2000). They also represent diverse sequences, structures and functions. Indeed, Luscombe *et al.* Luscombe and Thornton (2002) classified DNA–binding proteins across 54 structural–families. One important

problem where protein–DNA interactions play a fundamental role comprises the elucidation regulatory networks. Indeed, identifying proteins that bind DNA is the first step to decipher such networks. Subsequently, the identification of binding site on the protein structure will enable the protein to be docked to DNA facilitating the identification of the corresponding sites on the DNA. In sum, the following work describes a protocol that can identify the target proteins regulated by specific DNA–binding proteins expediting the construction of such regulation networks.

### 1.3.1 Test Case for Function Annotation

There is extensive annotation for DNA–binding proteins on both the sequence and structure level. For instance, the SwissProt Consortium (2007) contains about 20,000 annotated protein sequences known to bind DNA. Likewise, there is an exclusive database that tracks each DNA–binding protein having a solved structure Berman *et al.* (1992). Moreover, a number of DNA–binding proteins have been solved in complex with DNA White *et al.* (1998); Albright and Matthews (1998). And, there promises to be even more data pouring from high throughput techniques such as ChIP–chip experiments Buck and Lieb (2004).

Not only are there rich sources of data, this class of proteins has a distinct subcellular localization, the nucleus, and any protein residing outside the nucleus can be "safely" assumed not to bind DNA. Note, "safely" corresponds to the observation that only 6% of proteins are believed to bind DNA and it follows that while DNA–binding proteins residue outside the nucleus for a good portion of their lifetime, most of those found outside the nucleus will not interact with DNA. This makes the study of DNA–binding proteins ideal for a number of machine learning problems such as classification, multiple–instance learning and structured-learning. Moreover, the lessons learned studying DNA–binding proteins using machine learning techniques will transfer to the general elucidation of protein function.

### 1.3.2 Diagnostic Learning

Since DNA is negatively charged, a good portion of DNA–binding proteins are positively charged or bind using a positively charged surface. This characteristic alone is insufficient characterize a protein's ability to bind DNA. Indeed, a number of characteristics yet to be discovered compel a protein to bind DNA over another target such as membrane. Identifying important features that characterize a particular function is the first step to understanding the underlying mechanism that guides that function.

A machine learning model that generalizes the training data well (performs accurately on test data) comprises such important features and the relationships between them. Deriving information from a model built by a machine learning algorithm is referred to as Diagnostic Learning Liu *et al.* (2006). The knowledge extracted from such models allows

the user to interpret how the decision by the model was reached. For DNA–binding prediction, such knowledge gleaned from the model could lead to insights in protein/drug design or simply insights into how DNA–binding mechanisms have evolved.

### 1.3.3 Comparative Proteomics

Comparative proteomics is a concentration of comparative genomics Hardison (2003) where only the proteins (coding sequences of the genome) are compared. Moreover, adding information from another genome can help in gene prediction or functional annotation. Using comparative proteomics, we can answer a number of different questions depending on phylogenetic distance between genomes. For genomes with a long phylogenetic distance, we can garner broad insights into the characteristics of DNA–binding proteins. At moderate distances, we investigate the effect of negative selection on particular characteristics, i.e. the distinction between conserved and non-conserved. And finally, between similar genomes we tease out characteristics conserved or amplified due to positive selection.

## 1.4 Contributions

The primary contribution of this thesis is the development of a general machine learning strategy to replace standard sequence and eventually structure analysis techniques. This approach has been applied to function annotation of DNA–binding and membrane-binding proteins, recognition of the tertiary fold and identification of disease causing SAPs. The strength of this approach is that is thrives on all types of data and can be tailored to the problem or resolution as needed.

Another contribution of this thesis is a machine learning workbench that can handle a number of problems including classification and MIL. It provides a unified interface for several third–party classifiers as well as a number of wrapper methods to extend these classifiers to other problems. It also unifies validation and evaluation of the machine learning algorithms. A further contribution of this work is the development of a new boosting algorithm found in this workbench tailored to handle multiple–instance learning (MIL) problems. This algorithm is both efficient for large datasets and performs competitively with the current state–of–the–art in MIL.

In addition, this work utilizes diagnostic learning to investigate features of DNA–binding proteins invariant across genomes even at long genetic distances. These characteristics are vital for proteins to bind DNA. This work further investigates the evolution of DNA–binding characteristics where certain features are selected to allow better nucleotide discrimination and more robust mechanisms of regulation.

## 1.5 Overview of the Thesis

The thesis is divided into two parts. The chapters in the first part cover machine learning starting with background, introducing a new algorithm with experiments and concluding with an overview of the *malibu* machine learning workbench. The chapters in the second part cover applications of machine learning to a number of biological problems formulated as classification problems. This part wraps up with the focus on DNA–binding proteins, which provide a rich source of data for a number of machine learning problems.

The composition of this thesis precludes a single chapter covering previous work or a meaningful discussion. Alternatively, each chapter opens with the appropriate background, which ensures each chapter is fairly self contained. Each chapter also concludes with a its own discussion.

The chapters are organized as follows:

**Part I: Machine Learning**

**Chapter 2** introduces the relevant machine learning background. This chapter presents the formulation of a number of machine learning algorithms including classification, multiple–instance learning, probability estimation and importance–weighted learning. It also briefly introduces several machine learning algorithm implementations used in this work.

**Chapter 3** presents a novel modification of the AdaBoost algorithm. It details the previous work in modifying AdaBoost leading to the proposed modification. This modification allows boosting to work better on the instance–level for multiple–instance learning.

**Chapter 4** covers the *malibu* machine learning workbench. The *malibu* workbench serves as a unified interface for classifier implementations including third–party, native classifiers as well as wrapper algorithms that extend the native and third–party classifiers to other problems.

**Part II: Applications**

**Chapter 5** presents background and results in select biological problems including annotation of membrane-binding proteins, identification of disease causing non-synonymous single nucleotide polymorphisms and assignment of the tertiary protein fold.

**Chapter 6** presents background and results in a number of DNA–binding problems. This chapter introduces the current problems in function annotation with DNA–binding proteins. These include structure, sequence and residue annotation and their corresponding formulations.

**Chapter 7** presents results in a new field, comparative proteomics using DNA–binding proteins. Here, this work examines invariant features as well as the positive and negative selection of features important for DNA–binding.

# Chapter 2

# Machine Learning

One goal of this work is to cast biological problems as machine learning problems. This chapter introduces the background material in machine learning necessary to understand the contributions of this work. The first section of this chapter presents classification and introduces the classification algorithms used in this work. The second section introduces importance weighted classification, a generalization of cost-sensitive classification and a simple algorithm to make any classifier importance-weighted (and cost-sensitive). The third section introduces probabilistic regression and its connection to classification. It also presents three algorithms for obtaining accurate probability estimates from a classifier. The final section introduces multiple-instance learning and its connection to classification. This dovetails with the next chapter which presents a novel modification of AdaBoost for multiple-instance learning.

## 2.1 Classification

A supervised learning algorithm learns a function $f$ from a set of labeled examples, $(\vec{x}, y) \in S$, drawn independently and identically from a sample $S$ of some distribution $D$. This function maps an input $\vec{x} \in \mathbb{R}^d$ to an output $y \in Y$. One well understood supervised learning task is binary classification where $y \in \{0, 1\}$ and $\vec{x}$ has a fixed length. The binary classification problem attempts to find a classifier $e$ with the smallest error (see 2.1). An example of a binary classification problem is predicting whether a tumor is benign based on an MRI image where the image size is a fixed array of pixels and an expert labels the training images as cancerous or benign.

$$e(D, c) = Pr_{(\vec{x},y)\ D)}(c(\vec{x}) \neq y) \tag{2.1}$$

This problem is well understood; thus, there are a large number of algorithms that can successfully discriminate two classes based on a set of features. These classifiers fall into two main categories: classifiers and meta-classifiers. A meta-classifier builds an ensemble

of classifiers usually over some varying distribution of the training data in order to build a stronger learning algorithm.

### 2.1.1 Support Vector Machines

The Support Vector Machines (SVM) Cortes and Vapnik (1995) classifier belongs to the generalized linear classifier family. It attempts to find a hyperplane (in some higher dimensional space for non-linear problems) such that it has the largest possible margin between two groups of data. This is appealing because the largest margin should be robust to small changes in the training data. Moreover, only those examples that lie on the margin (called support vectors) are necessary to make a prediction; this gives the SVM model its sparse property. Indeed, current theory suggests Vapnik (1995) a large margin should lead to good generalization.

SVM formulates the classification problem as a quadratic optimization problem 2.2. Given labeled training data $\{\vec{x}_i, y_i\}$, $y_i \in \{-1, 1\}$, $x_i \in \mathbb{R}^d$, SVM attempts to find the hyperplane $\vec{w} \cdot \vec{x} + b = 0$, where $\vec{w}$ is normal to this hyperplane. Thus, SVM minimizes the Euclidean norm of $\vec{w}$ ($\|\vec{w}\|^2$) giving a pair of hyperplanes with the maximum margin.

The formulation in 2.2 is known as the soft-margin SVM. Specifically, this formulation does not require all the data to be classified with a margin of one. Instead, it allows some examples a little slack, which is measured by $\xi$. The hyper-parameter $C \geq 0$ trades *generalization of* with *fitting to* the training data.

$$\begin{aligned} &\min_{w,b} \tfrac{1}{2}\|\vec{w}\|^2 + C\sum_i \xi_i \\ &\text{subject to } y_n[\vec{w}^\top \Phi(\vec{x}_n) + b] \geq 1 - \xi_i \\ &\xi_i \geq 0 \end{aligned} \tag{2.2}$$

The SVM algorithm also utilizes a kernel, $\Phi(x_n)$, to map the data to a higher dimensional feature space where linear separation can be achieved on nonlinear data. In other words, the "right" kernel function allows SVM to effectively handle non-linear problems. There are a number of kernel transformations. For instance, a set of general kernels includes the gaussian, polynomial and sigmoid; each of these kernels represents a family of kernels such that the specific kernel is found by selecting the appropriate parameters. This selection is often done on a held-out validation set. In addition, kernels can also be constructed for a specific task, *e.g.* the string kernel Lodhi *et al.* (2002).

### 2.1.2 Decision Tree

A decision tree Quinlan (1986) learns a tree-structured model (see 2.1B) where every internal node represents a decision and every leaf a classification. This algorithm performs a greedy search using some loss function (usually referred to as an impurity function) to find the best axis-parallel split while partitioning the data space as it grows. In the simplest case, these splits are binary yielding two branches. However, there are cases

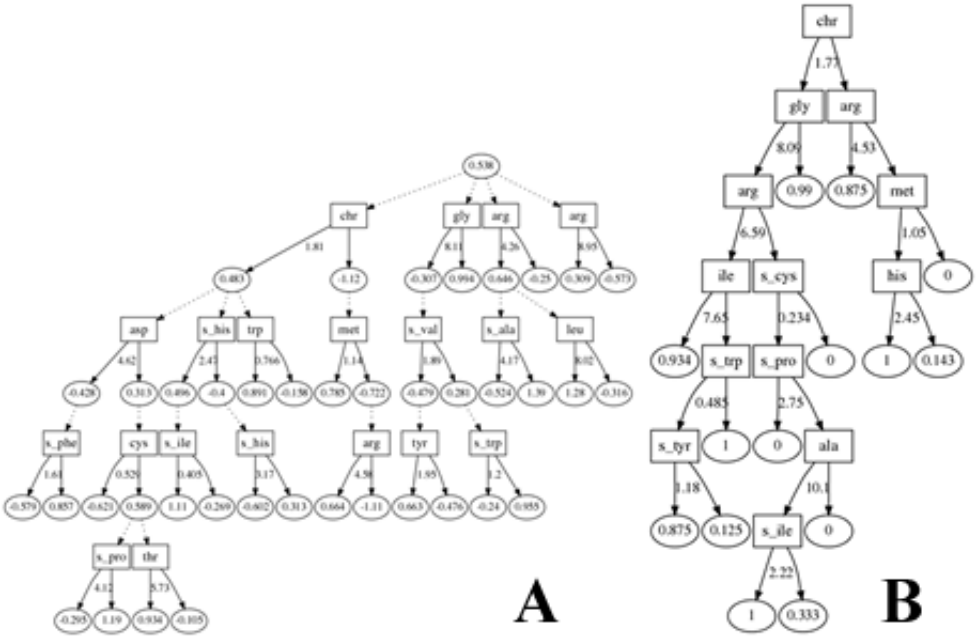


Figure 2.1: An example Alternating Decision Tree (A) and a Decision Tree (B).

when more splits are possible. For example, if an attribute is nominal (unordered) then it is common to create a branch for every possible value for the nominal attribute.

The decision tree suffers from two problems. The first problem is that disjunctive concepts are represented in different branches leading to the replication problem Pagallo and Haussler (1990). The second problem is that as splitting proceeds, the data associated with each node becomes smaller; this is known as fragmentation Pagallo and Haussler (1990). Eventually when the depth of the tree is large, there is little data associated with the deeper nodes and predictions (at deep leaves) become unreliable; such problems can be remedied by the meta-classifiers introduced later.

### 2.1.3 ADTree

The alternating decision tree (ADTree) algorithm Freund and Mason (1999) combines voted stumps and decision trees (see 2.1A). The AdaBoost algorithm Schapire and Singer (1999) (introduced later) is used to build the tree in a similar manner as first suggested by Kearns and Mansour Kearns and Mansour (1996). The final ADTree model is most similar to an option tree Buntine (1992) (voted decision tree), which as been shown to outperform single decision trees. One advantage of the ADTree algorithm is the improved comprehensibility, which has been studied in other algorithms by Craven Craven (1996) and others Domingos (1997); Margineantu and Dietterich (1997).

The interpretation of an ADTree differs considerably from the standard decision. That is, the ADTree makes a prediction by summing all the prediction (round in 2.1A) nodes transversed whereas a standard decision trees makes a prediction by following a *single* path to a prediction node. Thus, the standard decision tree leads to a rule of conjuctions forcing dependencies between attributes. The ADTree, however, represents both independent and dependent rules breaking the boolean analogy by combining independent rules using majority voting. Allowing branches to contain multiple independent rules allows the ADTree to represent more complex relationships than a standard decision tree Freund and Mason (1999).

### 2.1.4 Meta Classifiers

An ensemble of classifiers often outperforms a single classifier for three reasons. First, a learning algorithm searches the hypothesis space to find the best possible hypothesis. When the training data is small, a number of hypotheses may appear to be optimal. An ensemble will average the hypotheses reducing the risk of choosing the wrong one. Second, most classifiers perform a local search often getting stuck in local optima; multiple starting points provide a better approximation to an unknown function. Third, a single classifier may not be able to represent a the true unknown function. Some combination of these hypotheses, however, will be able to better represent this function.

### 2.1.5 Bagging

The bagging (bootstrap aggregating) Breiman (1996) algorithm creates an ensemble of classifiers by training each classifier on a random redistribution of the training set. Each random redistribution is generated by randomly drawing with replacement a number of examples equal to the size of the training set. This algorithm works well on learning algorithms like decision trees that are sensitive to the distribution of the data: having a large variation. In other words, it provides a method to extract robust aspects of the bootstrapped models by averaging out the noise. A unique advantage of bagging is the out-of-bag error, which gives a good estimate of how well the bagging classifier will perform on unseen instances and thus allows for efficient parameter tuning. The out-of-bag error is estimated by predicting each classifier in the bagging model with examples left out of their training sets.

### 2.1.6 Random Forests

The random forest Breiman (2001) algorithm is very similar to bagging. It creates an ensemble of classifiers by training each classifier on a random redistribution of the training set. Each random redistribution is generated by randomly drawing with replacement a number of examples equal to the size of the training set. Unlike bagging, however, random

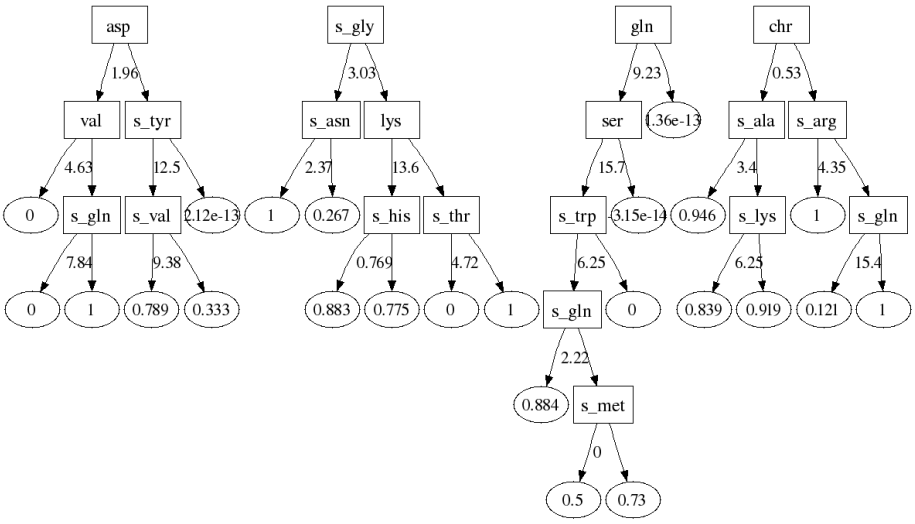


Figure 2.2: An example random forest classifier with four trees and an attribute subset size of five.

forests (see 2.2) also chooses a fixed size random subset the of attributes at each iteration; this subset is usually much smaller than the total number of attributes. The size of this attribute subset controls the error rate by trading strength for correlation. Specifically, a larger attribute subset increases both the strength and correlation between any two trees of the ensemble. Increasing the strength reduces the error rate while increasing the correlation increases it. An optimal subset size must be found using a held out test set or the out-of-bag error.

### 2.1.7 AdaBoost

The AdaBoost, Adaptive Boosting, Freund and Schapire (1996); Schapire and Singer (1999) algorithm creates an ensemble of *weak* classifiers by training each successive *weak* classifier on a distribution weighted towards past mistakes. Each *weak* classifier in the ensemble is weighted based on a transformation (2.3) of the total error, $\varepsilon$, of this classifier over the training set. The final prediction is a weighted sum over every classifier. The AdaBoost algorithm has proven particularly successful in its superior generalization.

$$\alpha = \frac{1}{2} \ln \frac{1 - \varepsilon_m}{\varepsilon_m} \tag{2.3}$$

## 2.2 Importance Weighted Classification

The importance weighted binary classification problem extends the binary classification problem by adding higher weight to more important examples. Each example forms a triple $(\vec{x}, y, i)$ where $i$ is the importance weight. The classifier minimizes the expected weighted loss (see 2.4) but the final solution is still a binary classifier. Note, cost-sensitive classification is a simple extension where each example is labeled depending on the class value.

$$e_w(D, c) = E_{(\vec{x},y,i)\ D)}[iI(c(\vec{x}) \neq y)] \tag{2.4}$$

### 2.2.1 Costing

The costing algorithm Zadrozny *et al.* (2003) extends a binary classifier to the importance weighted binary classification problem. Costing is an ensemble algorithm similar to bagging. At every iteration costing trains the base classifier over a sample (without replacement) of the original dataset. Each classifier is trained more often over important examples and less over unimportant. The final prediction is a majority vote over all the binary classifiers.

## 2.3 Probability Estimation

A probability estimate can provide a measure of the confidence in a prediction. Such estimates are necessary when the predicted output is not used in isolation but in the context of a greater system. This allows for another algorithm or person to make the final decision on the predicted output. Estimating probability estimates is a type of square error regression where the regressor attempts to minimize the square error (2.5) restricted to the range between zero and one.

$$e_r(D, h) = E_{(\vec{x},y)\ D)}[(h(\vec{x}) - y)^2] \tag{2.5}$$

### 2.3.1 Probing

The probing algorithm Langford and Beygelzimer (2005) extends an importance weighted binary classifier to a squared error regressor that outputs probability estimates. Unlike Platt and isotonic calibration (mentioned later), probing does not require a confidence value from the classifier. Instead, probing is based on the following observation: if a binary classifier predicts 1 for example $\vec{x}$ then it is predicting that the probability of class 1 given $\vec{x}$ is greater than the probability of class 0 given $\vec{x}$. Thus, it is possible to learn

classifiers which predict 1 if 2.6 is true and 0 otherwise by weighting the positive training examples to be $w$-times more costly than the negative ones.

$$D(y = 1|x) > \frac{1}{1+w} \tag{2.6}$$

The probing algorithm runs for a preset number of iterations and recursively refines (bins) the interval of greatest potential gain using the mean square loss function. The final prediction sums every classifier, which gives a zero-one prediction and uses the sum to index the corresponding probability bin (as estimated by the mean square error).

### 2.3.2 Platt Calibration

Platt scaling Platt (1999) attempts to fit the sigmoid function (see 2.7) to some measure of confidence produced by the classifier; in the case of SVM and Boosting, this would be the distance from the margin, which resembles a sigmoid curve. The form of this parametric model can be fitted using two parameters, $A$ and $B$, which are optimized over a validation set.

$$\hat{P}(1|\vec{x}) = \frac{1}{1+e^{Ac(\vec{x})+B}} \tag{2.7}$$

### 2.3.3 Isotonic Regression

When the shape of the mapping function is unknown, a non-parametric binning method can be used. In such a method, the training examples are sorted by the classifier output; usually this is estimated by three-fold cross-validation. Then, the sorted set is divided into a number of subsets (bins) of equal size. One such algorithm, isotonic regression Zadrozny and Elkan (2002) is a non-parametric form of regression (similar to binning) which assumes the function chosen is non-decreasing. A common algorithm for computing the isotonic regression is the pair-adjacent violators (PAV) algorithm Zadrozny and Elkan (2002).

## 2.4 multiple-instance Learning

The multiple-instance learning (MIL) problem is similar to classification where the training examples have incomplete label information. Instead of a label for each instance, there is a single label for a bag of instances. In the binary problem, a bag is labeled positive if at least one instance in the bag has a positive label otherwise the bag is labeled negative. The goal of multiple-instance learning is to predict the label of unseen bags or instances. Several techniques have been developed to solve this problem. The first class of algorithms, learning axis-parallel concepts, that popularized MIL was developed

by Dietterich in 1997 Dietterich *et al.* (1997) to solve the drug activity problem. In this problem, several drug molecules (bags) are known to bind or not to bind a particular target. Each drug molecule has several low-energy conformations (instances) but it is now known which conformation binds the target.

Similar to importance weighted classification and probability estimation, multiple-instance learning can be viewed as an extension of classification. Blum and Kalai Blum and Kalai (1998) present two reductions from multiple-instance learning (MIL) to classification. The first reduction simply views MIL as classification with one-sided noise. However, this reduction only uses only a single instance from each bag. The second reduction extends the Statistical Query model Kearns (1993) to MIL more efficiently using every instance in each bag.

## 2.5 Evaluation

The main theoretical question in classification is how well will a trained classifier perform on test data; in other words, how well will it generalize the training data to a new unseen set of examples. This is important for both evaluating the final learning model and selecting the best model from a collection of models. Model evaluation is broken into three parts. Firstly, validation algorithms partition a dataset into training and testing sets. Secondly, evaluation metrics measure the success or generalization of the learned model over a held-out test set. Finally, a plot measures the correlation between two metrics over a test set.

### 2.5.1 Validation

A number of algorithms exist to partition a dataset into training and testing sets. The motivation for many validation algorithms is to obtain an accurate performance estimate when limited data is available. This is, the more data available for training, the better the trained classifier performs. With limited data, leaving out too much will result in a pessimistic estimate of performance.

#### Holdout

The holdout technique is a fundamental method for evaluating the performance of a classifier; it also has the most theoretical support. In the hold out method, the dataset is split into two parts usually two-thirds for training and the left over one-third for testing.

#### Cross (validation)

For a dataset of small size, the $n$-cross-validation is demonstrably superior to holdout Goutte (1997). The cross-validation algorithm breaks a dataset into $n$ parts where $n-1$

parts compose the training set and the left out part composes the testing set. The process is repeated for each of the $n$ parts. An extreme case of cross-validation is called leave-one-out cross-validation where a single example of the training set is left out and the procedure is repeated for each example in the training set. Leave-one-out cross-validation is computationally expensive and 10-fold cross-validation is often used in practice Kohavi (1995).

#### Bootstrap

Another popular technique for small datasets is the bootstrap. Bootstrapping repeatedly analyzes subsamples of dataset where each subsample is drawn from the full dataset with replacement; this procedure is repeated anywhere from 50 to 1000 times. One of the main advantages of bootstrapping Blum *et al.* (1999) is that it estimates both the generalization error and confidence bounds.

### 2.5.2 Metrics

A metric is a single value that reflects some question asked about the classifier, ranker or regressor. Metrics are problem specific and here the focus is on binary classification problems. Since a number of the problems are skewed toward one class or another, we also consider ranking metrics which are less sensitive to class skew. Finally, regressions metrics measure the performance of probabilistic outputs by the learning algorithms.

#### Threshold

A confusion matrix or contingency table is a convenient means to tabulate statistics in order to evaluate a learning model. 2.1 holds the counts for true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN). Summing the columns gives the total predicted positive (PredP) and the total predicted negative (PredN); summing the rows gives total positive (NP) and total negative (NN). From the contingency table we can calculate a number of *threshold* metrics. These metrics are called threshold metrics because they depend on the threshold of prediction and/or class distribution.

The most common metric used in machine learning is the accuracy (or $1 - \text{Accuracy}$, the error) 2.8 where $y$ is the actual class value and $\hat{y}$ is the predicted class value. The accuracy answers the question, "What is the probability a classifier will predict the correct class label?". This probability is estimated with the fraction of correct predictions over the total number of predictions.

$$\text{Accuracy} = P(\hat{y} = y) \approx \frac{TP + TN}{N} \tag{2.8}$$

Since a binary classifier can make two types of errors, type I and type II errors (from statistics), there are corresponding metrics. The sensitivity measures the conditional

Table 2.1: Confusion matrix with marginals

| $N$ | $PredP^b$ | $PredN$ |
|---|---|---|
| $NP^a$ | $TP$ | $FN$ |
| $NN$ | $FP$ | $TN$ |

[a] Rows sum actual.

[b] Columns sum predicted.

probability an example is predicted positive given the example is positive. This is estimated by proportion of positive cases correctly predicted positive, 2.9. One minus the sensitivity is known as the false negative rate or type II error.

$$\text{Sensitivity} = P(\hat{y} = +|y = +) \approx \frac{TP}{NP} \tag{2.9}$$

The specificity measures the conditional probability an example is predicted negative given the example is negative. This is estimated by the proportion of negative cases correctly predicted negative, 2.10. One minus the specificity is known as the false positive rate or type I error. Note, the accuracy is a weighted average of the sensitivity and specificity.

$$\text{Specificity} = P(\hat{y} = -|y = -) \approx \frac{TN}{NN} \tag{2.10}$$

### Ranking

Threshold metrics have a singular limitation in that they depend on the class distribution or alternatively the threshold of prediction. Ranking metrics are insensitive to skew since they represent an average over all class distributions (or thresholds). One common ranking metric is the area under the receiver operating characteristic (ROC) curve (mentioned in the Plots section). The area under the ROC (AUR) measures the ability of the classifier to sort confidence rated predictions. For example, say classifier $C_1$ makes a single mistake predicting a negative instance to be positive with a confidence of 0.9. Now, classifier $C_2$ also makes a single mistake predicting a negative instance to be positive with a confidence of 0.51. Using accuracy, these classifiers are indistinguishable but using the area under the ROC, $C_2$ performs the best. In general, a classifier that produces less confident wrong predictions is preferable to one that does not.

#### Regression

A regression metric measures the agreement between the classifier output and the true value. Since this work is concerned with probabilistic output for a binary classification problem, the actual probability $p(i|x)$ for class $i$ given $x$ is 1 for the positive class and 0 for the negative class. One metric is the mean squared error (MSE), which measures (see 2.11) the expected value of the square error between predicted and actual probabilities. A MSE of zero indicates perfect accuracy whereas a MSE greater than or less than zero is only meaningful for comparative purposes.

$$\begin{aligned} \text{SE} &= E[(\hat{\theta} - \theta)^2] \\ &= \sum i(p(i|x) - \hat{p}(i|x))^2 \end{aligned} \tag{2.11}$$

### 2.5.3 Plots

The performance of the classifier cannot be entirely summarized with a single metric. Furthermore, a set of metrics cannot sum up classifier performance in every situation (*e.g.* different class distributions). Thus, a two-dimensional plot comparing the performance of a classifier using two metrics over a distribution of class ratios.

#### Receiver Operating Characteristic

The receiver operating characteristic (ROC) curve is useful for comparing classifiers and visualizing performance. It is particularly useful for domains with skewed classes and unequal classification costs. The ROC curve (see 2.3) plots the true positive rate (sensitivity) on the y-axis against the false positive rate (1 − Specificity) on the x-axis. A ROC curve is generated by sampling various class distributions from the original dataset or if the classifier produces confidence values, the threshold can be shifted from largest to smallest. The top right corner on the ROC curve corresponds to a classifier that only outputs positive predictions and the bottom left only negative predictions. One point in ROC space is superior to another if it is to the left and above another point. Any points along the diagonal correspond to a random classifier (worst possible case).

## 2.6 Summary

In this chapter, machine learning was reviewed including learning problems, specific algorithms, validation algorithms, metrics and plots. Each problem can be reduced (converted) to classification, a well understood learning problem, with a wrapper algorithm. This strategy of designing machine learning algorithms was first proposed by Beygelzimer *et al.* Beygelzimer *et al.* (2005a) and has both theoretical and practical algorithm design

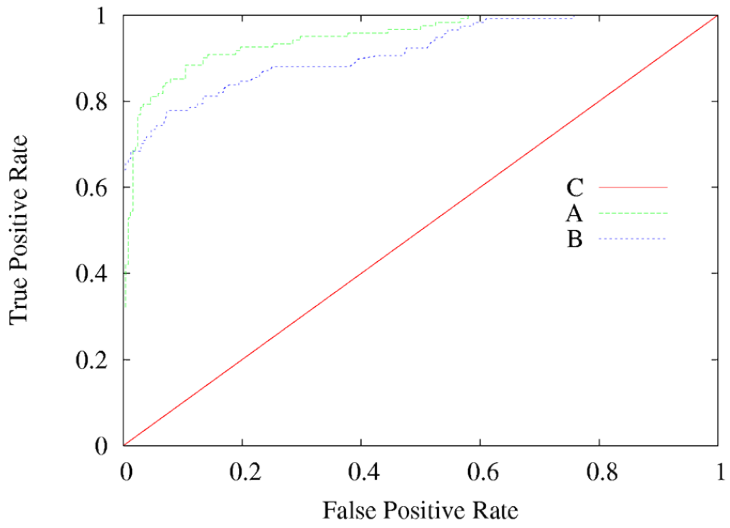


Figure 2.3: An example ROC plot comparing three classifiers A-C. Classifier A is the worst possible case giving random predictions. Classifier B dominates (does better than) classifier C while classifier A and B cross and A performs better over most of the plot.

advantages. For each problem with the exception of MIL, a set of algorithms was introduced. In the next chapter, a more detailed background of MIL will be given, along with the corresponding implementations and a new MIL algorithm is introduced.

# Chapter 3

# AdaBoost.C2MIL

This chapter introduces a novel boosting algorithm to tackle the multiple-instance learning (MIL) problem Langlois and Lu (2007b). This algorithm modifies AdaBoost to work well on both the bag and instance level. The first section introduces the multiple-instance learning problem and related algorithms to solve this problem. The second section reviews the boosting algorithm while introducing the new AdaBoost.MIL algorithm. It then presents an intuitive (yet unproven) modification that reduces AdaBoost.MIL to handle weak classifiers rather than weak MIL-learners called AdaBoost.C2MIL Langlois and Lu (2007b). The fourth section presents empirical results that demonstrate the superior performance of this new algorithm. The chapter wraps up with proposed future work.

## 3.1 Multiple-instance Learning

The multiple-instance learning (MIL) problem was first proposed by Keeler *et al.* for handwritten digit recognition Keeler *et al.* (1990). In this work, the position of a digit in the ZIP code is completely unknown. Keeler's approach, called Integrated Segmentation and Recognition (ISR), simultaneously learned the positions of the digits and parameters of a convolution neural network. Nevertheless, MIL did not become popular until Dietterich *et al.* Dietterich *et al.* (1997) resurrected MIL to solve the drug activity problem. In the drug activity problem, a set of drugs and their corresponding binding activity is known. However, each drug has many conformations and it is not known which conformation binds the target. Since it is more natural to represent a single conformation as a set of features rather than the drug itself, there is only weak label information available for the positive class. In other words, only the label on the drug is known not on the individual conformations. In the MIL problem, the conformations are called *instances* and the drug is called a *bag* where a bag can have any number of instances. In short, the MIL problem defines a *bag* as positive if at least one *instance* in the *bag* is positive otherwise the *bag* is labeled negative.

### 3.1.1 Axis-parallel

The first popular algorithm developed for MIL is axis-parallel concepts Dietterich *et al.* (1997). This algorithm searches for an axis-parallel hyper-rectangle (APR) to represent the target concept; the APR should contain at least one instance from each positive bag while excluding all the negative instances. Three algorithm variants were suggested to find such an APR. The first, *standard* algorithm finds the smallest APR that bounds all the instances from the positive bags. The second algorithm, *outside-in*, starts in a manner similar to the standard algorithm and then shrinks the APR excluding any false positives. The third algorithm, *inside-out*, starts from a seed point and then grows an APR to find the smallest rectangle that covers at least one instance of each positive bag and none from the negative. Empirically the last algorithm, *inside-out*, performs the best.

### 3.1.2 Diverse-Density

Maron and Lozano-Perez proposed diverse density (DD) Maron and Lozano-Pérez (1998) as a general framework to solve the MIL problem. The diverse density approach attempts to find a concept point close to at least one instance from every positive bag and far from every negative instance. The optimal point has the maximum diverse density, which is a measure of how many different positive bags have instances near the point and negative away from the point. The diverse density has a probabilistic derivation with no closed-form solution. Thus, a gradient ascent method must be used to search the feature space; this method is repeated for every instance in a positive bag as the starting point.

### 3.1.3 Other MIL Algorithms

A number of algorithms have been proposed to solve the MIL problem (the following is not comprehensive). Zhang and Goldman Zhang and Goldman (2001) use an EM (expectation maximization) approach to solve the diverse density problem (EM-DD). This approach models the knowledge of which instance determines the label of a bag as a set of hidden variables. Wang and Zucker Wang and Zucker (2000) introduce an adaptation of the k-nearest neighbor classifier to MIL, called citation-kNN. This is accomplished through a bag-level distance metric known as the minimum Hausdorff distance. Zucker and Chevaleyre Chevaleyre and Zucker (2001) introduced a modified C4.5 decision tree called C4.5-M to handle MIL. This tree uses an extended concept of the information gain (entropy) from labeled examples to labeled bags of examples.

Several approaches have been made to modify the support vector machines (SVM) algorithm. Andrews *et al.* Andrews and Hofmann (2003) reformulates the SVM optimization problem to maximize the usual instance margin jointly over the unknown instance labels, called mi-SVM. They also present an additional formulation that maximizes the bag margin, which is defined as the margin of the most positive instance in the the posi-

tive bag or margin of the least negative in the negative bag, called MI-SVM. Both these methods are mix-integer optimization problems; to this end Andrews *et al.* Andrews and Hofmann (2003) present a simple heuristic to handel such problems. Cheung and Kwok Cheung and Kwok (2006) also use a kernel method with a similar corresponding optimization problem; they solve their mix-integer optimization problem with a concave-convex procedure with known convergence properties. Gartner *et al.* Gärtner *et al.* (2002) introduce two new kernel functions for the MIL problem, normalized set kernel (NSK) and the statistic kernel (STK). Finally, Bunescu and Mooney Bunescu and Mooney (2007) combine both the multiple-instance kernels and altered constraints on the SVM object function.

Multiple-instance regression has also been introduced Ray and Page (2001). In Ray and Page Ray and Page (2001), the diverse density algorithm was extended to the real-valued case. Moreover, they also suggested an extension for the citation-kNN. Likewise, multiple-instance semi-supervised learning has also been presented Rahmani and Goldman (2006). Rahmani and Goldman Rahmani and Goldman (2006) present a graph-based semi-supervised method that uses a diverse density measure.

### 3.1.4 Reduction to Classification

It has been shown that MIL is a special case of binary classification Blum and Kalai (1998) or classification with positive class noise. In the classification setting, every instance takes on the label of its bag. While the reduction presented in Blum and Kalai Blum and Kalai (1998) uses only one instance from each bag, using every instance may help in specific domains. To this end, Ray *et al.* demonstrated that on many MIL problems Ray and Craven (2005), classifiers perform as well as if not better than MIL-oriented algorithms in terms of area under the ROC curve on the bag-level. However, a MIL counterpart of a specific learning algorithm usually performs better than that specific algorithm. Finally, they conjecture that algorithms with an MIL bias will probably function better in predicting instances rather than bags.

### 3.1.5 Multiple-instance Boosting

There have been several approaches that apply boosting to MIL. Each of these algorithms takes a different approach to modifying the AdaBoost algorithm. In short, the first approach constrains the weak learner to be an MIL-classifier called hyper-balls or hyper-rectangles. The second approach modifies the weight update of boosting using linear programming. The third approach boosts a generative probabilistic model. The fourth approach adds a MIL loss function to the AnyBoost framework Mason *et al.* (1999).

### Boosting Hyper-balls

The boosted hyper-ball algorithm presented in Auer and Ortner (2004) works on the bag rather than instance level. That is, the weights are assigned on the bag-level and the weak learner weight applies to a bag rather than an instance. Further, the weak learner is a multiple-instance learner rather than a classifier; it builds hyper-balls centered on positive bags. Two metrics are used to grow these hyper-balls including the 2-norm (hyper-ball) and the $\infty$-norm (hyper-cube).In the case of the hyper-cube, edges are extended until a negative instance lies on the boundary.

### Disjunctive Boosting

Disjunctive programming (DP) boosting Andrews and Hofmann (2003) extends the idea of linear programming (LP) boosting Demiriz *et al.* (2002) to the multiple-instance domain. The LP boost algorithm minimizes the 1-norm soft margin cost function found in support vector machines. In DP boosting, the linear programming problem is replaced by a disjunctive logic programming Balas (1985); Lee and Grossmann. (2000) problem (see 3.1). This allows the boosting algorithm to achieve a large margin for at least one instance in each bag. The training data is complied into a set of disjunctive constraints on $\alpha$. The problem can be efficiently solved through a convex relaxation of the constraints.

$$\begin{array}{l} \min_{\alpha,\xi} \sum_{k=1} {a_k}^n + C \sum_{k=1} {\xi_k}^m \\ \text{subject to} \quad (\alpha, \xi) \in Q \cap \bigcap_i \bigcup_{x \in X_i} H_i(x) \end{array} \tag{3.1}$$

### Generative Boosting

The generative model proposed in Xu and Frank (2004) uses a two state process similar to that used in diverse density Maron and Lozano-Pérez (1998). In the first stage, the class probabilities are calculated for individual instances in a bag. The second stage combines these probability estimates to assign a class label to a bag. Xu's generative model differs from diverse density in the second stage where all instances contribute equally the the classification of a bag. The boosting algorithm uses a modified loss function from the AnyBoost Mason *et al.* (1999) framework determining the weak learner weight uses numeric optimization. The boosted weights correspond to the bag rather than the instance similar to the boosted hyper-balls algorithm Auer and Ortner (2004).

### multiple-instance Boosting

Viola *et al.* proposed two new boosting variants to solve the MIL problem Viola *et al.* (2006). They derive these new boosting variants from the AnyBoost framework Mason *et al.* (1999) which views boosting as a gradient descent process. These new variants utilize an appropriate MIL cost function such as ISR (integrated segmentation and recognition)

and noisy OR. Following the AnyBoost approach, the weight on each example is the derivative of the cost function with respect to a change in the classifier score *e.g.* the noisy OR cost function (see 3.2). Likewise, the weak learner weight is determined using a line search to maximize the likelihood function (cost function in derivative of 3.2):

$$\frac{\partial \prod_i p_i^{t_i}(1-p_i)^{(1-t_i)}}{\partial y_{ij}} = w_{ij} = \frac{t_i - p_i}{p_i} p_{ij}, \tag{3.2}$$

where $p_i = 1 - \prod_{j \in i}(1 - p_{ij})$ and $t_i \in \{0, 1\}$ is the label of bag $i$. Intuitively, the weight for a negative instance is the same as that of the non-MIL AdaBoost framework *i.e.* all negative instances are equally negative. The weight for positive instance depends on both the bag and instance. That is, as learning proceeds and the bag approaches its target weight, the bag weight is reduced. Within the bag, examples that are predicted with a higher positive value are assigned higher weights; thus, these examples dominate subsequent learning.

## 3.2 AdaBoost.MIL

The Adaptive Boosting (AdaBoost) algorithm transforms a collection of weak classifiers into one strong classifier Freund and Schapire (1996). The original AdaBoost algorithm (AdaBoost.M1 for binary data also known as Discreet AdaBoost), is shown in Algorithm 1 using the classification loss in 3.3. This algorithm sequentially fits weak classifiers to a distribution controlled by a set of varying weights biased toward previous misclassifications. In other words, examples incorrectly predicted on a previous round receive greater weight on the next round and correctly predicted ones lower weight. The final AdaBoost classifier takes a weighted average over all the weak classifiers.

A more precise description of AdaBoost for binary classification shown in Algorithm 1 is stated as follows. A set of training data $(\vec{x}, y, w)$ is drawn from some distribution $D$ where $\vec{x}$ is a feature vector, $y \in \{-1, 1\}$ is a binary label and $w$ is an example weight distribution, initially set to the uniform distribution. The final boosted classifier $H(\vec{x}) = sign(\sum_{t=1}^{T} \alpha_t h_t(\vec{x}))$ composes a sum of weighted $\alpha_t$ weak learners $f(\vec{x}) \in \{1, -1\}$ where each weak learner produces a signed prediction. The AdaBoost algorithm trains weaker learners on a distribution weighted toward errors $w_{t+1} = \frac{w_t \cdot \exp(-\alpha_t y h_t(\vec{x}))}{Z_t}$ made by the previous weak learners $h_1(\vec{x}) \cdots h_t(\vec{x})$. This weak learner weight $\alpha_t = \frac{1}{2}\ln(\frac{1+e_t}{1-e_t})$ is chosen to minimize the upper bound on the training error, which is determined by the weights. Here, $e_t$ is the error of the weak learner on this round; for AdaBoost this error corresponds to the classification error 3.3 over the training set.

$$l_{t,i} = y_i h_t(\vec{x}_i) \tag{3.3}$$

In broad terms, the AdaBoost.MIL algorithm only differs in how the loss is calculated. Specifically, the MIL loss function (3.4) may be substituted without changing the general

**Algorithm 1** The AdaBoost Algorithm

Given: $(\vec{x}, y, w) \in D; \vec{x}_i \in X; y_i \in \{-1, +1\}$

Weak Learner: $h_t \in \{-1, +1\}$ ▷ A weak classifier

$w_{t=1,i} = \frac{1}{m}; i = 1, \cdots, m$ ▷ Uniform distribution

**for** $t = 1$ to $T$ **do**

$h_t \leftarrow Learn(D)$ ▷ Train classifier over training set

$e_t = \sum w_{t,(\cdot)} l_{t,(\cdot)}$ ▷ Estimate {1,-1} loss over training set

$\alpha_t = \frac{1}{2} \ln(\frac{1+e_t}{1-e_t})$ ▷ Calculate weight for learner

$w_{t+1,(\cdot)} = [w_{t,(\cdot)} \cdot \exp(-\alpha_t l_{t,(\cdot)})]/Z_t$ ▷ Update the weights for each example

**end for**

**Output:**

$$H(\vec{x}) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(\vec{x}))$$

AdaBoost algorithm. This, in turn, requires the weak learning algorithm to minimize the corresponding loss function such that AdaBoost.MIL requires a weak MIL learner rather than a standard (weak) classifier. Given $i \in j$ is the index of the $i$th example (instance) in bag $j$, ($t_j \in \{0, 1\}$ is the label on the bag and $b_t(\bar{x}_j) \in \{-1, 1\}$ is a bag level prediction. The MIL loss function in 3.4 measures loss on both the bag and instance level. That is, if the bag is negative $(1 - t_j)$, the loss is similar to the original AdaBoost loss (3.3) on the instance level. When the bag is positive, the instances contribute in an all-or-nothing fashion: if the bag is correctly predicted, then every instance is correct otherwise every instance is incorrect and the weights contribute accordingly. Note, the weights are assigned on the instance level rather than the bag-level. This adds the extra requirement that the weak MIL learner be able to handel instance-level weights.

$$l_{t,j,i} = (1 - t_j)(y_i h_t(\vec{x}_i)) + t_j b_t(\bar{x}_j) \;\; \forall i \in j \tag{3.4}$$

It is simple to prove that this new error function does not violate the gradient descent view of boosting. Using the proof from Schapire and Freund Schapire and Singer (1999), given $u_i \in \{-1, +1\}$, $Z$ can be represented as:

$$Z = \sum_i w_i e^{-\alpha l_{(\cdot)}} = \sum_i w_i (\frac{1 + l_{(\cdot)}}{2} e^{-\alpha} + \frac{1 - l_{(\cdot)}}{2} e^{\alpha}) \tag{3.5}$$

The right hand side of this equation (3.5) can be minimized by analytically choosing $\alpha$ giving:

$$\begin{aligned} \alpha &= \tfrac{1}{2} \ln(\tfrac{1+e_t}{1-e_t}) \\ e_t &= \textstyle\sum l_{(\cdot)} \end{aligned} \tag{3.6}$$

While we have proved AdaBoost.MIL works under the gradient descent framework first proposed by Friedman Friedman *et al.* (2000) and later extended by Mason *et al.* Mason *et al.* (1999), this only proves AdaBoost.MIL is a leveraging algorithm Duffy and Helmbold (1999). Proving this algorithm is strictly a boosting algorithm requires proving it has the (PAC) boosting property Duffy and Helmbold (1999). We save this for future work.

## 3.3 AdaBoost.C2MIL

So far we have only shown a simple modification of AdaBoost to handel the MIL loss function. Now we would like to propose an additional modification that would allow AdaBoost.MIL to boost weak classifiers rather than weak MIL-classifiers. In other words, we want to reduce multiple-instance learning to classification using the AdaBoost algorithm. This presents several practical and theoretical advantages. First, we only require a weak classifier whereas our boosting algorithm builds a strong MIL-learner. Second, since

classification is well understood, we only need to prove that our modification performs within a specific error bound. Third, we can use any of the available classifiers to model the underlying distribution.

The proposed modification simply reweights the examples in a positive bag after the normal AdaBoost weighting scheme such that an example predicted positive will have a higher weight and one predicted negative a lower weight. The within bag weighting scheme forces the weak learner to focus on these "positive" instances in later rounds. This idea is similar to one proposed in Viola *et al.* (2006) except I propose a closed-form solution. In short, the following weight update in 3.7 is used after the normal AdaBoost.MIL update and is normalized such that the weight of the bag remains the same as before this update.

$$w_{t+1,i} = \frac{w_{t,i} \cdot \exp(\hat{\alpha}_t y_i h_t(\vec{x}_i))}{\hat{Z}_t} \quad \forall i \in j \quad \text{and} \quad t_j = 1 \tag{3.7}$$

By maintaining the bag weight computed after the first weight update, we argue that the overall loss function is not changed. That is, for a positive bag, the weight update only depends on prediction of the entire bag and thusly only the entire bag weight (see in bold 3.8). If the bag weight does not change during this second update, then the second update does not change the loss function.

$$\begin{aligned} e &= \textstyle\sum_j \sum_{i \in j} w_i[(1-t_j)(y_i h_t(\vec{x}_i)) + \boldsymbol{t_j b_t(\bar{x}_j)}] \\ &= \begin{cases} \sum_{i \in j} w_i y_i h_t(\vec{x}_i), & t_j = 0 \\ \sum_{i \in j} \boldsymbol{t_j b_t(\bar{x}_j)}, & t_j = 1 \end{cases} \end{aligned} \tag{3.8}$$

This is equally true for the normalization since it only depends on the total bag weight for a positive bag (see 3.9). Thus, the new loss function only affects the weak learning algorithm.

$$Z = \sum_j \sum_{i \in j} w_i e^{-\alpha[(1-t_j)(y_i h_t(\vec{x}_i)) + \boldsymbol{t_j b_t(\bar{x}_j)}]} \tag{3.9}$$

This leaves only the update itself. So far, I shown step by step how to modify AdaBoost to handel multiple-instance learning without changing the core features of the algorithm. Currently, I have only an intuitive argument for choosing the $\hat{\alpha}$ to be the $\alpha$ estimated by AdaBoost. Specifically, the weight of the entire positive bag is adjusted by $\exp(-\alpha_t l_{t,(\cdot)})$, however, only one instance in that bag is required to be positive in order for the bag to be predicted positive. Thus, choosing to reweight the instances in the bag by $\exp(\alpha_t l_{t,(\cdot)})$ emphasizes the (most likely) positive instance(s) and deemphasizes the (most likely) negative instances. This drives AdaBoost to focus a weak classifier (rather than a weak MIL-classifier) on the smaller core true positives in later rounds. Thus, all that remains is to prove this weighting scheme minimizes the MIL-error when using a classifier or to find a better scheme that can be proved to do the same. In the next section, I will present results that support the current weighting scheme.

Table 3.1: Statistics of the MIL datasets

| | Attributes | Example | Positive | Negative | Bags | Positive | Negative |
|---|---|---|---|---|---|---|---|
| Elephant | 230 | 1391 | 762 | 629 | 200 | 100 | 100 |
| Fox | 230 | 1320 | 647 | 673 | 200 | 100 | 100 |
| Tiger | 230 | 1220 | 544 | 676 | 200 | 100 | 100 |
| Musk1 | 166 | 476 | 207 | 269 | 92 | 47 | 45 |
| Musk2 | 166 | 6598 | 1017 | 5581 | 102 | 39 | 63 |

## 3.4 Experiments with MIL-Boost

Multiple-instance learning has been applied to a number of problem domains including text categorization, fold identification, content-based image retrieval and drug design. Here, I will first introduce specific problem domains and their corresponding datasets used to evaluate the new MIL algorithm. Then, I will investigate the ability of various classifiers to handle multiple-instance learning as a one-sided noise problem including AdaBoost. Subsequently, I will compare AdaBoost on decision trees with AdaBoost.C2MIL on the same trees. Finally, I will compare AdaBoost.C2MIL (on decision trees) to other multiple-instance learning algorithms.

### 3.4.1 Problem Domains

One problem domain where MIL is a natural formulation comprises the drug activity problem Dietterich *et al.* (1997), which motivated the current popularity of MIL research. In the drug activity problem, one wishes to assess which of a set of potential drugs will strongly bind a target. Each drug has a dynamic three dimensional structure moving between a number of different conformations. A drug that shows activity must have a conformation that strongly binds the target whereas a drug that shows no activity has no such conformation. However, the ability of specific conformation to bind is unknown. So, each drug has an associated label indicating whether it interacts with the target but also forms a bag of instances that correspond to the various conformations. Previous work has focused on one dataset, the Musk dataset, which has two variants, Musk1 and Musk2. While both variants have the same number of bags, Musk2 has a greater number of instances associated with each bag 3.1.

Another domain that benefits from the MIL representation is content based image retrieval (CIR) *e.g.* Maron and Lozano-Pérez (1998); Zhang *et al.* (2002) where the task involves finding images that contain objects of interest. In other words, the image corresponds to a bag and segments of the image an instance and the object of interest is assumed to be in at least one segment for positive bags. In the following experiments,

Table 3.2: Compares classifiers over the Musk datasets

| Dataset | Algorithm | Accuracy | Sensitivity | Specificity | AUR |
|---|---|---|---|---|---|
| Musk1 | SVM | 82.7 | 96.6 | 68.2 | 90.5 |
| | C4.5 | 73.0 | 92.8 | 52.4 | 74.3 |
| | CR-AB[a] | 85.9 | 96.2 | 75.1 | 96.0 |
| | RF[b] | 81.1 | 87.4 | 74.4 | 89.4 |
| | ADTree | 78.3 | 93.6 | 62.2 | 84.4 |
| Musk2 | SVM | 74.0 | 95.9 | 60.5 | 88.0 |
| | C4.5 | 57.9 | 88.7 | 38.9 | 65.7 |
| | CR-AB[a] | 76.5 | 87.7 | 69.5 | 90.8 |
| | RF[b] | 73.5 | 74.4 | 73.0 | 78.9 |
| | ADTree | 60.8 | 84.6 | 46.0 | 71.1 |

[a] CR-AB: Confidence-rated AdaBoost

[b] RF: Random Forests

the learner attempts to separate three types of animals, tiger, elephant and fox from background images Andrews and Hofmann (2003) broken down in 3.1.

### 3.4.2 Classifiers

Since multiple-instance learning (MIL) can be reduced to classification with one sided noise, I start by investigating the performance of several classifiers over two MIL datasets.

While there has been previous work making such a comparison Ray and Craven (2005), it is limited to only a few classifiers (not including AdaBoost) and only presents results in terms of area under the ROC curve (AUR).

In this work, I focus on classifiers including support vector machines (SVM), the popular C4.5 decision tree, Real (Confidence-rated) AdaBoost on Decision Trees (CR-AB), Random Forests on C4.5 (RF) and the Alternating Decision Tree (ADTree). Each of these classifiers is introduced in Chapter 2 and further expanded upon in Chapter 4. The parameter selection is unique to each algorithm type. Firstly, the C4.5 decision tree was grown with default values. Secondly, the number of iterations for ADTree and CR-AB is set by 5-fold cross-validation. Finally, the random forests use the out-of-bag error to select tree size using 200 trees for parameter selection and 1000 to build the best model.

Comparing Musk1 and Musk2 in 3.2, it is clear that each classifier performs worse on the Musk2 dataset. This is consistent with greater class noise in the positive examples since each positive bag now has close to 20 instances whereas for MUSK1 the average is about 3 instances. However, what is counter-intuitive is the discrepancy between Random Forests (RF) and boosted trees (CR-AB). Specifically, AdaBoost has been shown to per-

Table 3.3: Comparing AdaBoost.C2MIL on the bag-level

| Algorithm | Dataset | Accuracy | Sensitivity | Specificity | AUR |
|---|---|---|---|---|---|
| AdaBoost | Elephant | 76.5 | 100. | 53.0 | 92.9 |
| | Fox | 61.0 | 96.0 | 26.0 | 69.8 |
| | Tiger | 75.0 | 98.0 | 52.0 | 92.0 |
| | MUSK1 | 87.0 | 100. | 73.3 | 98.8 |
| | MUSK2 | 78.4 | 92.3 | 69.8 | 93.4 |
| AdaBoost.C2MIL | Elephant | 79.0 | 59.0 | 99.0 | 94.2 |
| | Fox | 57.0 | 15.0 | 99.0 | 67.9 |
| | Tiger | 76.5 | 55.0 | 98.0 | 91.4 |
| | MUSK1 | 78.3 | 68.1 | 88.9 | 87.5 |
| | MUSK2 | 82.4 | 64.1 | 83.7 | 88.2 |

form poorly in domains with higher class noise whereas random forests are usually robust to such noise. Here, the opposite it true. Moving from Musk1 to Musk2, the difference between the two classifiers grows from 6% to 12% in terms of AUR. In sum, the results show that AdaBoost is competitive if not superior on these MIL tasks and thus will form a good baseline for latter comparisons.

### 3.4.3 Compare Bag-level: AdaBoost

The results in 3.3 compare AdaBoost to the new algorithm, AdaBoost.C2MIL on the bag-level. Both of these algorithm boost the same decision trees and both are run for the same number of iterations, 1024. Note, a bag is predicted positive if at least one instance is positive and negative otherwise. Since the classifier predicts instances, one expects a disproportionately high sensitivity to specificity. These algorithms are tested over five datasets: three from CIR and two from drug activity. At first glance, AdaBoost performs better in terms of AUR on all but one dataset.

However, the sensitivity and specificity illustrate the advantages of AdaBoost.C2MIL. That is, on every dataset AdaBoost.C2MIL consistently performs well in predicting negative bags. On the image content retrieval task, it consistently reaches over 90% specificity. Likewise, the accuracy remains competitive to AdaBoost on every task but Musk1. Given the small number of instance per bag in Musk1 it is not surprising a classifier does better on this task. In sum, AdaBoost.C2MIL performs well in terms of accuracy on most of the MIL problems.

Table 3.4: Comparing AdaBoost.C2MIL on the instance-level

| Algorithm | Dataset | Accuracy | Sensitivity | Specificity[a] | AUR |
|---|---|---|---|---|---|
| AdaBoost | Elephant | 83.0 | 85.2 | 80.3 | 90.3 |
| | Fox | 62.3 | 66.5 | 58.3 | 66.8 |
| | Tiger | 77.8 | 77.9 | 77.7 | 85.6 |
| | MUSK1 | 85.5 | 89.4 | 82.6 | 93.8 |
| | MUSK2 | 85.7 | 72.7 | 79.1 | 82.6 |
| AdaBoost.C2MIL | Elephant | 51.1 | 11.0 | 99.7 | 80.6 |
| | Fox | 52.2 | 0.02 | 99.9 | 58.6 |
| | Tiger | 60.8 | 12.5 | 99.7 | 78.6 |
| | MUSK1 | 65.8 | 24.2 | 97.8 | 72.1 |
| | MUSK2 | 85.4 | 0.07 | 99.7 | 80.9 |

[a] Specificity is the only reasonable metric.

### 3.4.4 Compare Instance-level: AdaBoost

The results in 3.4 compare AdaBoost and AdaBoost.C2MIL on the instance level. While the accuracy, sensitivity and AUR are not reasonable metrics (true label on instance in positive bag is unknown), they do give some insight into the underlying predictions. For example, AdaBoost.C2MIL optimizes an error criterion that allows it to accurately predict the negative instances only scoring less than 99% on the Musk1 dataset. Similarly, since AdaBoost minimizes the error, it consistently does "well" in every other metric. Comparing the bag and instance level for AdaBoost.C2MIL on the Elephant dataset, a 11% sensitivity on the instance level leads to a 59% sensitivity on the bag level. On the same dataset, AdaBoost yields 66% on the instance level leading to 100% sensitivity on the bag level. Thus, the new algorithm is able to classify 60% of the bag using only one sixed of the instances in positive bags. In other words, the new algorithm should work much better on the instance level than AdaBoost.

### 3.4.5 Compare MIL Algorithms

The results in 3.5 compare AdaBoost.C2MIL to several popular MIL-classifiers including diverse density (DD) Maron and Lozano-Pérez (1998), multiple-instance linear regression (MI/LR) Ray and Craven (2005), and two kernels for support vector machines: NSK and STKGärtner *et al.* (2002). This comparison is done in terms of area under the ROC (AUR) complied from other work Ray and Craven (2005).

Compared to the state-of-the-art MIL algorithms, AdaBoost.C2MIL performs very well. On three of the five datasets, it performs significantly better than its counterparts.

Table 3.5: Compares AdaBoost.C2MIL to MIL using AUR

| | C2MIL[a] | DD[b] | MI/LR[c] | SVM-STK[d] | SVM-NSK[e] |
|---|---|---|---|---|---|
| Musk1 | 87.5 | 89.5 | 86.7 | **93.7** | 92.4 |
| Musk2 | 88.2 | **90.3** | 87.0 | 89.2 | 86.6 |
| Tiger | **91.4** | 84.1 | 89.0 | 70.5 | 84.8 |
| Elephant | **94.2** | 90.7 | 93.3 | 85.6 | 91.5 |
| Fox | **67.9** | 63.1 | 63.3 | 61.8 | 52.5 |

[a] C2MIL: AdaBoost.C2MIL

[b] DD: diverse densityMaron and Lozano-Pérez (1998)

[c] MI/LR: Multiple-instance linear regression Ray and Craven (2005)

[d] SVM-STK: An MIL Statistics kernel for support vector machines. Gärtner *et al.* (2002)

[e] SVM-NSK: An MIL normalized set kernel for support vector machines. Gärtner *et al.* (2002)

Moreover, it performs comparably on Musk2 and only performs worse on Musk1 (the least MIL dataset). This is especially encouraging because AdaBoost.C2MIL is far more efficient than all of the algorithms.

## 3.5 Discussion and Future Work

There are many MIL algorithms that have not been benchmarked on freely available datasets and for those algorithms that have been benchmarked only a single metric is given. Thus, it would be useful to have a wider benchmark. This would ideally include several other popular datasets and the creation of some new datasets.

One such dataset would involve identifying binding sites where only sequence information is available. This experiment could be performed on two levels. First, using structures and known binding sites, the first dataset that gives accurate instance level labels could be created. Then, this dataset could be expanded to include sequences where the binding sites are unknown. One of the main motivations for AdaBoost.C2MIL was to develop an efficient algorithm that could give accurate instance level results *i.e.* accurately identify binding sites on sequences.

Finally, further theoretical work will investigate the properties of the new algorithm. First, the final weight modification must be analyzed theoretically to prove that it translates a weak classifier into a weak MIL-classifier giving a reduction from MIL to classification. Second, this algorithm seems to maintain the PAC-boosting property. Proving it has the property will lend this algorithm much credence in the machine learning community.

# Chapter 4

# The *malibu* Workbench

This chapter presents the development of a machine learning workbench undertaken in parallel to the rest of this work Langlois and Lu (2007d). Unless otherwise noted, every machine learning result reported in this thesis was achieved using this workbench. With a number of available machine learning workbenches, why create another one? "The need for open source software in machine learning" is carefully laid out in the following work Sonnenburg *et al.* (2007). In short, the Journal of Machine Learning Research at Massachusetts Institute of Technology has started a new publication track for open source machine learning workbenches in order to provide more incentive to develop such open source software. The advantages of open source software for machine learning research includes:

1. Reproducibility of results, leading to fair comparisons
2. Uncovering problems in current algorithms
3. Building on existing resources

A machine learning workbench itself gives the user a unified interface for a number of different algorithms and takes over mundane tasks *e.g.* validation and/or parameter tuning. Further, having a number of machine learning algorithms available allows the user to find the algorithm that best suits the situation. Indeed, the no-free-lunch theorem Wolpert and Macready (1995); Wolpert (2001) states that a general algorithm will always perform worse than one specifically designed for some problem; however, on average over all problems they will perform the same. Thus, it pays to carefully select an algorithm for the target problem domain.

This chapter is structured as follows. In the first two sections, I will introduce existing tools and the corresponding motivation to develop a new machine learning workbench. In the third section, I will cover the general design of the workbench. In the fourth section, I will explore the user-interface as well as the expected format for the dataset

and the output formats for the predictions and model. In the fifth section, I will focus on the problems solvable by this workbench. In the sixth section, I will cover the available validation methods, metrics and plots. Finally, I will summarize the key points in this chapter.

## 4.1 Existing Tools

The following tools are confined to popular open source supervised machine learning toolkits that support a number of different algorithms. The following workbenches are grouped by programming language since this largely defines key features including speed, usability and portability of the workbench.

### 4.1.1 Java-based Tools

The advantages of Java-based tools include portability and usability. That is, a Java-based tool can run on any operating system that has the interpreter installed. Since there is an interpreter available for most operating systems, Java tools are very portable. Likewise, Java provides an extensive library for graphical user interfaces. Two of the three workbenches listed below leverage this advantage into well designed user interfaces.

The disadvantages of the following Java-based tools includes efficiency, remote usability and workflow. For example, a Java-based implementation generally runs 2–3 times slower than a native implementation; this does not take into account garbage collection on data intensive applications. Likewise, the following programs provide an interface for running the learning algorithms on a local machine. While it is possible to script these algorithms for remote usage, the corresponding design makes this a laborious task. Finally, they support running a single experiment whereas the average researcher will probably need to run many experiments. Thus, without good workflow, distributing multiple experiments to many machines shifts an enormous amount of work to the user.

#### WEKA

WEKA Witten and Frank (2005) is a comprehensive and popular machine learning workbench. While its strength is in classification, it also handles multiple-instance learning, regression, association rules and clustering. Moreover, there are a number of side development projects extending WEKA to spectral clustering, meta-learning, datamining, Bayesian networks, to name a few. Moreover, there are also extensions for grid computing, Grid WEKA Pérez *et al.* (2005), and parallel computing, WEKA-parallel Celis and Musicant (2002).

WEKA also defines two graphical user interfaces (GUIs). The first GUI, called the Experiment Environment, allows the user to create, run, modify and analyze experiments

in a single run. The second GUI, called the WEKA Explorer, defines the following tasks:

1. Preprocess: Choose and transform a dataset.
2. Classify: Train and validate schemes for classification and regression.
3. Cluster: Learns unsupervised partitions of the dataset.
4. Associate: Learns associations rules.
5. Select Attributes: Choose more relevant features.
6. Visualize: View 2D plot of the data.

#### Rapid Miner (formerly YALE)

Rapid Miner aims to be a rapid prototyping framework for machine learning tasks Mierswa *et al.* (2006). It meets several requirements to accomplish this goal. First, it is very flexible in its methods for preprocessing and data analysis. In other words, it contains a good deal of preprocessing methods and a mechanism to easily incorporate new ones. Second, it is able to process a number of data types from time series to text data without effort by the user. Third, it provides an extensive toolkit for evaluation and optimization. Fourth, it has an intuitive user interface. This system is driven by an XML scripting language that allows any number of nestable operators for an experiment. These XML files can be created by their graphical user interface. Note, a good portion of Rapid Miner algorithms are derived from WEKA.

### 4.1.2 C/C++-based Tools

The advantage of C/C++ based tools is efficiency in terms of memory and speed, which is important for CPU and memory intensive machine learning algorithms. Indeed, a nice graphical user interface is not necessary when running an application on a remote machine. However, such workbenches often include additional tools coded in scripting languages like Python (see Orange), which can be used to build a nice user interface while connecting to a faster C/C++ algorithm implementations. The biggest disadvantage is the range and scope of available machine learning algorithms due to the greater difficulty in developing a tool in C/C++.

#### MLC++

MLC++ provides an extensive set of algorithms for machine learning Kohavi *et al.* (1996). First, it provides what they call inducers, a set of rule miners and classifiers. Second, it provides what they call wrappers and hybrids, a set of algorithms that extend the basic inducers. For example, two wrapper methods include accuracy estimators (cross-validation,

hold, or bootstrap) and features selectors. In addition, certain wrappers can also perform dataset transformations. MLC++ includes utilities to estimate generalization of the inducers and build graphical representations of corresponding learning curves. This workbench has no graphical user interface and requires all parameters to be set using environmental variables; this strange design makes scripts non-portable.

#### Orange

Orange has a set of core C++ objects with a python interface in a scriptable environment Demšar *et al.* (2004). The graphical interface includes an orange canvas, which allows for a visual widget assembly stage. The Python modules for data manipulation include sampling, filtering, scaling and discretization. Likewise, it codes classification and regression methods as well as wrappers for boosting, bagging and probability estimation. Finally, it has a large set of evaluation methods including holdout schemes and classification metrics. The main disadvantage of Orange is the limited number of available algorithms.

#### TORCH

Torch comprises an object-oriented paradigm implemented in C++ Collobert *et al.* (2002). It is broken down into four broad classes: Dataset, Machine, Trainer, and Measurer. The dataset reads and holds the examples, the machine comprises the core algorithms, the trainer selects the best parameters based on the performance estimated by the Measurer. Torch concentrates on gradient machines *i.e.* multi-layer perceptrons as well as other algorithms that can be trained by gradient descent. It also includes support vector machines and distribution models (*e.g.* Gaussian mixture model or hidden Markov models) trained by expectation-maximization. The main disadvantage of Torch is the limited number of available algorithms and limited user interface.

#### LEGMA

Lemga stands for learning models and generic algorithms Li (2001). It comprises several learning models including generic optimization training models. The learning models consist of neural networks, perceptrons, support vector machines, pulses and the decision stumps. These base learners can be extended by aggregation methods including bagging, AdaBoost, CGBoost, and LPBoost. Finally, the generic algorithms are provided including gradient descent, line search and conjugate gradient methods. The main disadvantage of Lemga is the limited number of available algorithms and limited user interface.

### 4.1.3 Matlab-based Tools

The advantages of Matlab includes fast prototyping, some portability, and ease of leverage multiple CPUs. Matlab is a much simpler language than C/C++ or Java with an extensive

mathematical function library. It easier to connect two algorithms, write a new one or change an existing one. In addition, since Matlab is an interpreted language, programs written in Matlab can run on any operating system Matlab can. Finally, Matlab code can be easily run on multiple CPUs when compared to Java or C/C++. The main disadvantage of Matlab-based tools includes both the expense of a Matlab license and efficiency of Matlab coded scripts, which can be more intensive in terms of memory and CPU than C/C++ algorithms.

#### The Spider

The Spider machine learning library Weston *et al.* (2003) was developed to handle reasonably large unsupervised, supervised or semi-supervised machine learning problems. This workbench defines a set of plug and play objects that can be connected to solve a wide range of problems. While this approach allows fast prototyping, it also places the burden of learning Matlab on the user. In addition, it also includes algorithms for evaluating results, model selection and performing statistical tests. For example, it supports the Wilcoxon test of statistical significance as well as the corrected resampled t-Test; this in addition to the rather extensive set of statistical tests already built-in to Matlab. Finally, for a large-scale analysis, several core objects have been written to work with datasets in terms of memory and speed.

### 4.1.4 Delphi-based Tools

Delphi was designed to perform rapid application development and it is popular with both desktop applications and commercial database tools. This high-level language is often compared to an object-oriented Pascal. The main disadvantage of Delphi is the lack of open source software available and free tools for development.

#### TANAGRA

TANAGRA Rakotomalala (2005) codes a set of algorithms to handle a wide set of problems including supervised learning, cluster, factorial analysis, parametric/non-parametric statistics, association rules, and feature selection. This workbench serves as an experimental platform that removes from the user responsibility for data management. The interface comprises three parts including a stream diagram as a tree view, set of nodes as operators and components (learning algorithms and utilities), and an HTML output report. Similar to the previous tools (with the exception of WEKA using third-party extensions), TANAGRA works only on a local machine with no remote interface.

## 4.2 Motivation

*malibu* Langlois and Lu (2007d) was designed to be a simple, effective machine learning workbench to handle supervised learning problems; it was developed to handle several deficiencies in previous workbenches. First, none of the previously described workbenches were designed to effectively work alone in a remote console environment. To this end, the *malibu* interface aims to emulate many standard Linux command-line programs and also supports configuration files. This allows a user familiar with Linux to easily run *malibu* algorithms; since most algorithms run for significant amounts of time, a remote Linux machine is a likely environment. Second, these workbenches also do no provide a method to quickly prepare results for publication. Indeed, *malibu* works seamlessly with GNUPlot (http://www.gnuplot.info/) and Graphviz (http://www.graphviz.org/) to build customizable plots and model illustrations. Moreover, it provides metric tables in several formats including LaTeX HTML and tab-delimited. Third, the previous workbenches also do not support mechanisms to reproduce results. That is, the *malibu* output file contains all the information to calculate any metric or plot as well as a configuration file in order to repeat the experiment. This saves the researcher time in copying configuration files or keeping notes. Finally, most of the previous workbenches do not utilize third-party code whereas *malibu* leverages many robust third-party implementations including C4.5 and LibSVM. Thus, the user can trust the algorithms provided are the best possible implementations.

## 4.3 Design

*malibu* defines a set of core classifiers that form the foundation of the workbench. In order to strengthen or extend the classifiers to new domains, *malibu* also defines a set of wrappers. Unlike other workbenches, the classifiers and wrappers are selected at compile time giving maximum efficiency during run-time. *malibu* utilizes C++ meta programming, which has been made popular by projects like Boost (www.boost.org) and the The Matrix Template Library (http://www.osl.iu.edu/research/mtl/). The template mechanism permits high performance modular programming.

The object organization of the machine learning algorithms in *malibu* is shown in 4.1. The Tree class exemplifies an atomic or core classifier at the root of the hierarchy. At the same time, the AdaBoost and Bagging class templates serve as wrappers to improve the results of the core classifier. Moreover, both of these wrappers can be nested to create another meta-classifier. Thus, a great number of learning algorithms can be built from a few base classifiers and wrappers at compile time.

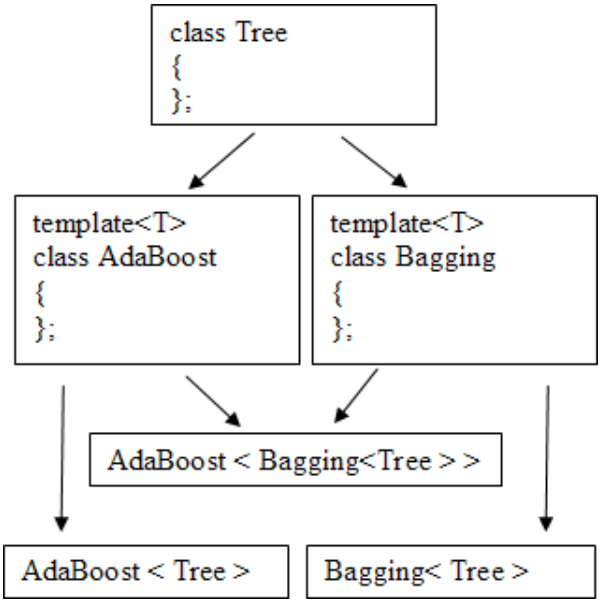


Figure 4.1: This figure shows the general organization of the *malibu* objects.

## 4.4 Interface

The interface for a machine learning workbench comprises the following parts: algorithm parameters, prediction output, model output and dataset input. Each of the files read into or written out of *malibu* can be zipped or unzipped using the *bzip2* compression algorithm.

### 4.4.1 Dataset Formats

A dataset generally comprises a header followed by rows of attribute values. Frequently, each row is prefixed with one or more label identifiers and one attribute represents the class attribute (see 4.2). By default, *malibu* detects a header by checking if every attribute on the first line is a string label; it also allows the user to force the first line to be a counted as a header or ignored. Also by default, *malibu* assumes the first column is a label; this can be changed by setting the number of expected labels. Finally, the first column after the label is assumed to be the class attribute; this can similarly be changed to any column.

Each learning algorithm in *malibu* supports this dataset format and the following preprocessing options. First, the user can set which label should be considered positive; a multi-class dataset can be reduced to binary by listing the classes that should be positive (all others will be negative). Second, the user can set a particular label (or label substring) to indicate a grouping of examples into bags for multiple-instance learning or special

| Label | Class | Attribute(1) | Attribute(2) | Attribute(3) | Attribute(4) |
|---|---|---|---|---|---|
| Example1 | 1 | 92.3 | 91.6 | 86.8 | 88.6 |
| Example2 | 0 | 57.5 | 65.0 | 65.0 | 55.0 |
| Example3 | 0 | 98.7 | 96.1 | 90.6 | 94.4 |
| Example4 | 1 | 93.6 | 84.6 | 88.1 | 65.5 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

Figure 4.2: An example dataset in the standard *malibu* format.

classification tasks.

### 4.4.2 Prediction Output

The prediction output file uses a flat file labeling system to ensure ease in later parsing. Every line with the exception of the config file (appended to the end) is prefixed with a '#' and a two letter code (see 4.3). The first section generally holds summary statistics for each dataset (training and/or testing). This is following by the type of validation and any parameter tuning output (if specified). The main body consists of a prediction header and the individual predictions. The prediction header holds the learning algorithm name, the validation type, and the training set name. This is followed by the prediction rows, which consist of an example label, the class label, the prediction and a prediction threshold. The last part of each prediction output file is a configuration file and since '#' is a config file comment character, the prediction output file can serve as a configuration file so that the experiment can be repeated. Note, this does not explicitly save the states of the random number generator.

### 4.4.3 Model Output

The current model format for each *malibu* component (built-in or not) is a text-file format. This allows the user to peruse the model file. *malibu* also supports a graph output format for a subset of (mostly tree-based) algorithms. This allows the user to construct a nice illustration of the learned model using a number of third-party tools including Graphviz (http://www.graphviz.org/). This functionality was used to build the ADTree models shown in Chapter 2 and Chapter 6.

### 4.4.4 Parameters

*malibu* supports a number of methods for parameter input. First, it supports command-line parameters, which are subdivided into two types: master arguments and flag arguments. A majority of the parameters are controlled by flag arguments; those of the form

```
#LS     Filename: elephant.csv
#LS     Bags: 200
#LS     Examples: 1391
  ⋮       ⋮
#ETVA   C4.5 | 5-fold Cross-validation | elephant.csv
#SPVA   0000  1   0.28   0
#SPVA   0007  1   0.37   0
  ⋮       ⋮   ⋮    ⋮     ⋮
#SPVA   0114  0  -0.31   0
#SPVA   0108  0  -0.39   0
  ⋮       ⋮   ⋮    ⋮     ⋮
```

Figure 4.3: An example dataset in the standard *malibu* format.

`-flag value`. Depending on the algorithm, master arguments may also be used; these do not have a flag associated with them where order is used indicate the corresponding parameter. *malibu* also supports a configuration file for flag arguments; it contains flag value pairs and comments (usually descriptions of the program arguments). Since the learning algorithms often have many arguments and many of these are shared, they can also output a portion of the configuration file. For instance, a learning algorithm can output all arguments associated with parsing a dataset or all arguments associated with the particular algorithm. Moreover, any number of configuration files can be used as master arguments for a learning algorithm.

## 4.5 Problems

This section covers machine learning problems that *malibu* can currently solve, which fall under the auspice of supervised learning. In other words, each problem considers a set of labeled training examples used to builds a model, which can predict the label accurately on unseen examples. Each problem has a large set of associated algorithms since no single classifier performs best on a problem domain.

### 4.5.1 Classification

A binary classifier learns a model from a set of labeled examples and outputs a binary prediction of the label on some example. *malibu* incorporates a number of classifiers both third-party and built-in. The third-party classifiers include LIBSVM Chang and Lin (2001), Cover Tree kNN Beygelzimer *et al.* (2006), INDTree Buntine and Caruana

(1991) and C4.5 Quinlan (1993, 1996). The built-in classifiers include the Willow decision tree and ADTree Freund and Mason (1999). It also includes a number of built-in wrapper methods including Bagging Breiman (1996), Subagging Buhlmann (2003), AdaBoost Freund and Schapire (1996) and Confidence-rated AdaBoost Schapire and Singer (1999); Friedman *et al.* (2000).

### LIBSVM

The LIBSVM Chang and Lin (2001) third-party implementation distinguishes itself from other implementations by using the sequential minimal optimization (SMO) decomposition method. Since quadratic optimizers can only efficiently handle problems up to a certain size, a decomposition method must be employed to make large problems tractable. That is, an optimizer updates the weight vector iteratively. However, the problem may be too large to be solved at each iteration. A decomposition method updates only a subset of examples per iteration called the working set. In SMO, only two examples are updated at each iteration leading to slow convergence. A new implementation of LIBSVM (2.8) utilizes second order information to improve convergence Fan *et al.* (2005).

The LIBSVM library supports a number of support vector machine algorithms. This includes two algorithms for support vector classification called C-SVC and nu-SVC. In a sense, both of these algorithms are equivalent with different parameter ranges. It also supports two support vector regression algorithms similar to the classification called epsilon-SVR and nu-SVR. It also provides an semi-unsupervised algorithm for density estimation called one-class SVM.

Finally, there are a number of kernel functions available in LIBSVM. The kernel functions allow the SVM algorithm to map non-linear problems into linear space where the large margin separation is achieved. These kernel functions include the gaussian, polynomial, linear and sigmoid. Most published works rely on the gaussian kernel followed by the polynomial and linear.

### Cover Tree (kNN)

The Cover Tree third-party implementation Beygelzimer *et al.* (2006) implements the k-nearest neighbor algorithm. It uses a new data structure that allows a query of the k-nearest neighbor in $O(log(n))$ assuming a fixed intrinsic dimensionality. Moreover, the space and query time are $O(n)$ under no assumptions. In practice, the speed improvements are considerable. Finally, this algorithm is written as a generic C++ template and thus extensible to other domains. For example, it can easily support weighted instances.

### INDTree

The INDTree Buntine and Caruana (1991) third-party package implements thirteen tree algorithms, three graph algorithms and supports mix and matching various tree growing

strategies to build other tree types. The package includes several popular tree algorithms including classification and regression trees (CART) and a C4.5 variant called C4. In addition, it also supports trees grown with the minimum message length (MML) principle. MML is the formal restatement of Occam's Razor in information theory terms; the simplest best trees are grown. It also includes a lookahead decision tree that searches possible future pathes before an attribute is split. Finally, it supports an option tree implementation, a predecessor to the ADTree algorithm (covered later).

#### C4.5

The C4.5 Quinlan (1993, 1996) is another third-party implementation of the decision tree induction algorithm. First, it supports both nominal and real attributes for both binary and multiple splits. For the nominal splits, it supports a subsetting algorithm that groups nominal attributes into two groups and a multi-split algorithm that splits on each possible value. In addition, it implicitly handles missing attributes. Second, C4.5 determines the best split using information gain (or gain ratio for nominal attributes), which is based on entropy. Finally, C4.5 utilizes a efficient form of pessimistic pruning that only requires the tree to be grown twice as compared to 5 or 10 times for cross-validation.

#### Willow

The Willow built-in implementation of the decision tree induction algorithm is similar to C4.5; it supports real, nominal, subset, and missing attributes. It also utilizes a range of impurity functions including misclassification, Gini Index, Entropy, KMD Kearns and Mansour (1996), and least-squared error. The tree growth is controlled by depth, minimum number objects to split and confidence in a split; this can be tuned using cross-validation. The trees can also produce either probabilities or confidence values making them ideal for boosting. Finally, it allows for random subsets of attributes of a fixed size; this allows Willow to serve as a base tree learner in the random forest algorithm.

#### ADTree

The ADTree Freund and Mason (1999) built-in implementation uses a boosting algorithm to induce a tree using Willow stumps (one-level decision trees). The name, alternating decision tree, refers to the structure of the constructed tree comprising alternating levels of prediction and decision nodes. Due to its improved comprehensibility, this implementation of ADTree can be represented in a graph format (*e.g.* Graphviz) and thusly can be represented as a graph in an image format.

### 4.5.2 Meta Classification

A meta-classifier takes a committee of weak classifiers creates a strong classifier. Specifically, the following classifiers work well on decision tree algorithms. The meta-classifiers are broken into two main groups: sample aggregating (*e.g.* Bagging) and boosting (*e.g.* AdaBoost) algorithms. While both algorithms work to reduce the variance of a weak learning algorithm, only boosting algorithms also reduce the bias.

#### Sample Aggregating

The sample aggregating (built-in) meta classifier builds a strong classifier by training some weak classifier on samples drawn from the training set. Both sampling with replacement (for the Bagging Breiman (1996) algorithm) and without replacement (for the Subagging Buhlmann (2003) algorithm) are supported. This algorithm can be used to construct a bagging, subagging or random forest classifier. It also supports parameter selection using the out-of-bag error by averaging the error of all the weak classifiers tested on the left out portion of their individual training sets; this is a very efficient form of parameter selection.

#### Boosting

That AdaBoost Freund and Schapire (1996) built-in implementation takes a weak classifier and creates a committee of weighted "weak" classifiers to form a strong classifier. This algorithm supports both sampling (with and without replacement) for weak learners that do not accept weights as well as those algorithms that support weights. It also supports Confidence-rated boosting when either the weak learner produces the correct confidence value or probability estimate that can be transformed by the boosting algorithm. Using Willow trees, the AdaBoost algorithm can be extended to Discreet AdaBoost Freund and Schapire (1996), Real (Confidence-rated) AdaBoost Schapire and Singer (1999) and Gentle AdaBoost Friedman *et al.* (2000).

### 4.5.3 Importance Weighted Classification

Importance weighted classifiers learn a function biased toward more important examples. Any algorithm that accepts weights is an importance weighted algorithm. If the algorithm is unweighted, then the costing reduction Zadrozny *et al.* (2003) can be used to extend the classifier to importance weighted classification; note, costing is a variation of the sample aggregating implementation listed above where weights determine the likelihood that an example will be included in the sample. Cost sensitive classification is one simple extension of importance weighted algorithms where weights are assigned depending on the class of the example.

### 4.5.4 Probability Estimation

A probability estimation classifier produces probability estimates for predicted examples. There are two types of probability estimation algorithms; calibration algorithms Zadrozny and Elkan (2002); Platt (1999) that convert classifier confidence value to probability estimates and the Probing Langford and Beygelzimer (2005) algorithm converts a discrete output to a probability estimate. The built-in probing implementation follows that outline in Langford and Beygelzimer (2005) extending an importance weighted classifier to a probability estimator.

The built-in calibration algorithms use validation (usually 3-fold cross-validation) to estimate conditional probability estimates. The Platt calibration Platt (1999) implementation follows the improved strategy of Lin *et al.* Lin *et al.* (2007); it adds a theoretical convergence guarantee and overcomes numerical difficulties. The isotonic calibration Zadrozny and Elkan (2002) implementation follows the pseudo code suggest by Ayer *et al.* Ayer *et al.* (1955).

### 4.5.5 Multiple-instance Learning

In multiple-instance learning (MIL), examples are grouped into bags where the bag not an individual example has a label. *malibu* uses a string label on an example to group examples into bags. Every classifier in *malibu* can perform MIL (where instances receive the label from the bag) as covered in Chapter 2 of this thesis. *malibu* also supports a new MIL version of the AdaBoost algorithm called AdaBoost.C2MIL covered in Chapter 3.

A side-benefit of the MIL bag grouping is that such groupings can be used in some classification domains to obtain a better estimate of generalization. For example, in protein binding site classification each residue is predicted whether its binds DNA or not. When a validation algorithm like cross-validation is used all the instances are sorted without considering to what protein they belong. Thus, grouping amino acids on the protein (bag) level gives a stronger estimate of how the classifier will perform on a held out protein (rather than an amino acid) in a more realistic setting.

## 4.6 Evaluation

Evaluation of the classifier is used to both evaluate the generalization ability of the final model and to select the final model from a set of models. The evaluation is broken up into three parts: validation algorithms, metrics and plots.

### 4.6.1 Validation

Validation algorithms determine the best way to partition the dataset into a training and a test set. The choice of algorithm depends on the size of the dataset and allowable

computational time. That is, for smaller the datasets, less examples are available for training and the resulting performance is a pessimistic estimate of the actual performance; thus, using a validation algorithm that leverages more training data at the expensive of increased computational time will give a better estimate. There are three basic types of validation methods: partitioning, sampling and online.

### Partitioning

A partitioning method splits the dataset in two parts one for testing and one for training. There are three types of partitioning methods supported in *malibu* : cross-validation, holdout and inverted holdout. The proportion of training examples used in training and testing is set using the $n$-fold parameter. The $n$-fold or number of folds corresponds to the number of (roughly) equally sized partitions of the dataset. Note, the partitioning methods support stratification *i.e.* maintaining the dataset class distribution in each partition. For holdout, $n - 1$ folds are used in training and 1 fold for testing. Cross-validation repeats holdout until every partition is used once for testing. Finally, inverted holdout uses one fold for training and $n-1$ for testing. Generally, cross-validation is used on small, holdout on medium and inverted holdout on large datasets; this is mainly for computational reasons trading accuracy with training set size, which is directly related with computational efficiency.

### Sampling

The sampling methods draw a subsample of the data for training and uses the left out portion for testing; this process is performed repeatedly. The alpha parameter controls the portion of training and testing examples for sampling without replacement. If the alpha parameter is set to zero (for *malibu* only), sampling with replacement is used. In sampling with replacement, approximately two-thirds of the dataset is used for training on each iteration.

One related validation technique supported only by bagged classifiers and random forests (a special case of bagging) is the out-of-bag error. Since these techniques sample the dataset, the left out examples can be used to estimate the performance of the classifier. This validation technique can only be used for model selection.

### Online

Another validation scheme, progressive validation Blum *et al.* (1999), finds its roots in online learning. In online learning, the learning algorithm is presented with an unlabeled example and is then asked to predict the label; the algorithm is then given the actual label of the example to update is current model. Since this is a well studied and understood problem domain, the progressive validation scheme has all the theoretical advantages of holdout while using (on average) one-half the number of examples for testing. This

algorithm works as follows: the dataset is partitioned into two sets, the learning algorithm is trained over one portion and subsequently is asked to predict the label of an example drawn from the second set. After each example is tested it is then added to the training set and the learning algorithm is retrained over this new set. While this strategy is computationally demanding, it is very attractive for small datasets.

### 4.6.2 Measures: Metrics and Graphs

After the learning algorithm produces an output file containing a set of predictions (see 4.3), the *malibu* evaluate program can be used to measure the performance (in terms of generalization) of the learner. The program supports a number of metrics including all those presented in Chapter 2. These metrics can be formatted in a number of layouts including in a HTML table that can be pasted directly into any HTML editor like Google Docs and a LaTeX table. It also outputs a number of graphs with a header and coordinates formatted for GNUPLOT. Indeed, this command can be piped directly to GNUPLOT *e.g.* `evalf out | gnuplot`. If the above supported formats do not work, it also supports a template parser that permits user defined formats.

## 4.7 Summary and Future Work

*malibu* is a supervised learning workbench focused on solving tasks related to classification. The workbench is divided into two main parts: learning and evaluation supporting a number of validation algorithms, metrics and plots for evaluating a learned model over a test set. The design allows the user to select/build a learning algorithm at compile time yielding the best possible algorithm in terms of efficiency for run-time.

In future work, we first plan to add more algorithms including neural networks, logistic regression, naïve Bayes and restricted Boltzman machines. These algorithms represent a diverse set of methods and eventually *malibu* will combine these algorithms to build a meta learner (*e.g.* stacking) to generate the best possible prediction for any learning task. *malibu* is also focused on convenience *i.e.* getting the results into a paper with as little effort as possible. To this end, we plan to add an summary output file in LaTeX and HTML that describes the dataset, learning algorithm and performance in a comprehensive manner. Finally, the current trend in computing is grid computing and thusly *malibu* will be equipped with a user interface that will handel scheduling algorithm tasks across multiple CPUs on multiple distributed computers.

# Chapter 5

# Biological Problems

This chapter introduces a general framework that can be used to understand the underlying mechanisms of certain genetic diseases; that is, those caused by mutations leading to single nucleotide polymorphims (SNPs). While SNPs can cause genetic disease, they are also fundamental in human genetics such that: they contribute to traits that make an individual unique, they can be used to track inheritance, and they underly susceptibilities to common ailments such as heart disease or arthritis. Our understanding of SNPs is still relatively new and more applications lie on the horizon *e.g.* SNPs could explain variation in drug responses among individuals. One type of SNP occurs in a region of DNA coding for a protein called a nonsynonomous single nucleotide polymorphism or a single amino acid polymorphism (SAP); such SNPs are important in that their effect on the resulting protein structure forms an explicit testable model.

This framework can be broken down into three distinct yet important biological problems. First, protein fold assignment represents a classic multi-class classification problem where a classifier attempts to assign a protein sequence to one of a large set of three-dimensional protein structural folds. Second, the disease causing single amino acid polymorphism problem delineates the identification of genetic variants that cause disease from those that do not. Finally, in the membrane-binding annotation problem, a classifier attempts to distinguish three-dimensional protein structures that bind membrane from other protein structures.

The overall framework uses the following protocol as shown in 5.1. The first step involves building accurate three-dimensional models from a protein sequence; the protein fold assignment problem introduced in the first section deals with the particularly hard cases. Then, based on the structure the function can be determined. Finally, potential coding mutations (SAPs) in this protein can be mined to determine disease association. From this computer simulated data, we can determine potential candidates for disease association studies or gain understanding of known disease-associated proteins.

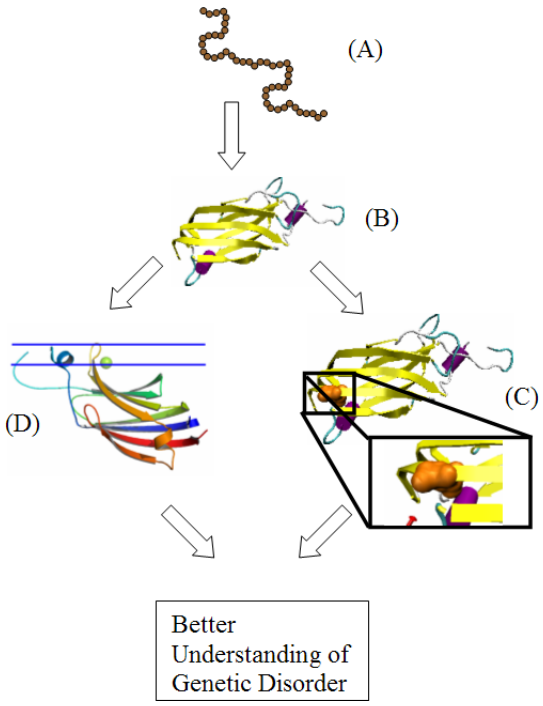


Figure 5.1: The following problems represent key steps in understanding the mechanistic cause of a genetic disorder. The protocol starts with a sequence (A) of unknown function and attempts to model the structure (B). Then a mutation is proposed and the modelled mutant structure is predicted as to whether it could cause a disease (C). In parallel, the function is determined of the sequence (D). The final disease mechanism is proposed by this protocol.

## 5.1 Protein Fold Assignment

Protein fold assignment is a central problem in molecular biology and accordingly many techniques have been developed to detect the structural class of a primary sequence. The structure of a protein provides a rich set of features, which can be used to determine the function of the protein. However, the corresponding experimental methods such as x-ray crystallography and nuclear magnetic resonance (NMR) spectroscopy are very time consuming (on the order of months to years) and expensive. Moreover, while there are thousands of protein structures in the PDB, there are still millions of sequences yet unsolved.

The structural class (or fold) of a protein is classified into a hierarchical system called SCOP (structural classification of proteins) Murzin *et al.* (1995). This classification is manually curated and covers thousands of folds. There are four approaches to solving the structure from the sequence: homology modelling Karplus *et al.* (1998); Sanchez and

Sali (1997), generative models such as threading Skolnick and Kihara (2001); McGuffin and Jones (2003); Xu and Xu (2000), denovo folding Bonneau *et al.* (2001); Kihara *et al.* (2001) and discriminative techniques Dubchak *et al.* (1999); Jaakkola *et al.* (2000); Leslie *et al.* (2002a); Hou *et al.* (2003); Weston *et al.* (2005). A discriminative method has been demonstrated superior Leslie *et al.* (2004); Liao and Noble (2002) (in the binary case) given sufficient data and appropriate sequence representation; this is further improved by semi-supervised approach Weston *et al.* (2005); Kuang *et al.* (2005); Rangwala and Karypis (2005). Most of these techniques extend sequence alignment with support vector machines (SVM) Liao and Noble (2002); Jaakkola *et al.* (2000) or perform the sequence comparison in a kernel Leslie *et al.* (2002b,a, 2004); Saigo *et al.* (2004). Another approach is to use sequence-structure correlationsHou *et al.* (2003, 2004) or motifs Saigo *et al.* (2004) as features. However, the ASTRAL database holds more than 1000 fold classes Brenner *et al.* (2000); Murzin *et al.* (1995) and dealing with this multi-class problem is particulary challenging.

There are generally two approaches to solving a multi-class classification problem: reduction to a binary problem Dietterich and Bakiri (1995); Allwein *et al.* (2000) or generalize a binary classifier Crammer and Singer (2001) (and for a nice review Rifkin and Klautau (2004)). The first method is generally preferred to the second for mainly computational reasons. One multi-class to binary reduction is one-versus-all (OVA) where a single class is positive and the rest negative; this is repeated for each class. Another approach is one-versus-one (OVO) that trains all pairwise classifiers $N(N-1)/2$; the final output is a vote Allwein *et al.* (2000). Likewise, error-correcting output codes (ECOC) represent multiple classes with a binary vector called an output code. The class output is derived from a vector of binary predictions, which is measured against a vector corresponding to the actual class output Dietterich and Bakiri (1995); Allwein *et al.* (2000). All of the methods discussed so far can be improved using an importance weighted binary classifier. For example, the OVA reduction is further improved by weighted OVA Beygelzimer *et al.* (2005b). Likewise, the ECOC reduction is also further improved SECOC (sensitive ECOC) Langford and Zadrozny (2005). Finally, the most recent results suggest the filter tree style tournament on an importance-weighted binary classifier is currently the best available Beygelzimer *et al.* (2007).

The application of binary classifiers extended to fold recognition has been extensively studied. Dubchak *et al.* Dubchak *et al.* (1999) developed one of the initial multi-class protocols for classifying a sequence into one of many folds. This initial work used neural networks as the classifier and was later improved upon using the support vector machines classifier Ding and Dubchak (2001). This approach was later generalized by Yu *et al.* Yu *et al.* (2003) using a jury voting scheme that combines several SVMs over different feature representations. These approaches were later improved using error correcting codes to combine binary SVM classifiers Ie *et al.* (2005); Melvin *et al.* (2007).

Our own work utilized a novel secondary structure descriptor (using predicted sec-

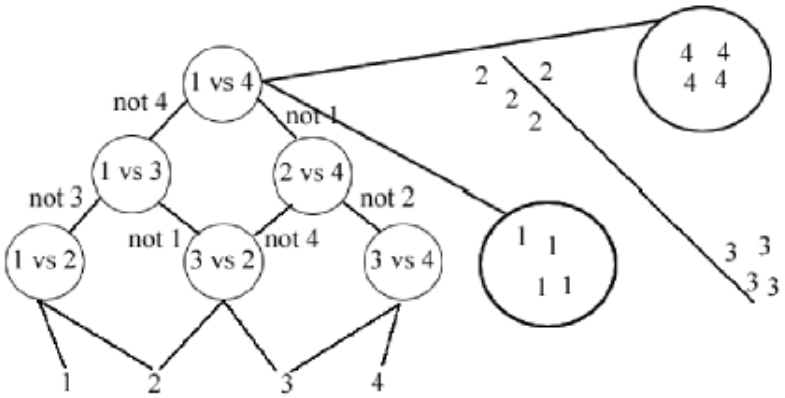


Figure 5.2: A graphical representation of the decision directed acyclic graph voting system.

ondary structure) to describe protein sequences Langlois *et al.* (2004, 2006). We used support vector machines extended by the DDAG reduction to multi-class classification first suggested by Platt *et al.* Platt *et al.* (1999). Unlike previous work, we compare directly with the current state-of-the-art in protein fold recognition, threading, and find our method works comparably.

### 5.1.1 Method

As previously described in both Chapter 2 and Chapter 4, we used the LibSVM Chang and Lin (2001) implementation of the SVM classifier Cortes and Vapnik (1995). However, the multi-class extension was developed before the *malibu* workbench. As described in Langlois *et al.* (2006), we implemented the decision directed acyclic graph (DDAG) multi-class to binary reduction Platt *et al.* (1999). This method trains (see 5.2) $\frac{N(N-1)}{2}$ classifiers similar to OVO yet only performs $n-1$ predictions. The parameters for the SVM classifier were chosen using 5-fold cross-validation and the performance is measured on a held-out test set.

#### Dataset

The state in the fold recognition problem comprises a set of sequences, which are classified into different categories (folds) according to the SCOP system Murzin *et al.* (1995). SCOP classifies a protein according to a hierarchical system breaking down into class, fold, super family, and family. In previous work Dubchak *et al.* (1999), it has been shown that class level classification can be achieved with high accuracy. Similarly, using homology-modelling techniques, super family level and family level recognition has not proven hard for current methods. To this end, we have focused on assigning protein sequences on

the fold level. Given the difficulties in reconstructing Dubchak's 27-fold dataset (missing identifiers and reclassified sequences), we opted to create our own, larger dataset. Note, while Dubchak *et al.* provide an online dataset, this is composed of the features, not the original sequences. The ASTRAL40 Brenner *et al.* (2000) database was used to ensure no example had more than 40% identity to another. Moreover, only folds with no less than 20 examples are taken to ensure sufficiently large testing and training sets for both accurate and significant results, respectively. The final dataset consists of 53 folds were the training and testing sets consist of 2013 and 530 examples, respectively. That is, for each of the 53 folds, the test set has exactly 10 examples whereas the training set has no less than 10 examples.

#### Features

Two descriptors were constructed to represent a protein sequence. The first descriptor simply comprises the amino acid composition (a count normalized to sequence length) of the protein sequence. The second descriptor encodes the composition of specific secondary structure topologies. That is, this descriptor starts by counting the number of secondary structure elements, helices, sheets and loops. It, then, counts the number of helices followed by sheets, the number of helices followed by helices and so on. This was extended to topologies of size eight. The secondary structure was assigned using predictions from PsiPred McGuffin *et al.* (2000).

### 5.1.2 Results

The results reported measure performance using confidence and accuracy, which are really multi-class extensions of the precision and recall (see Baldi *et al.* (2000) for an overview of multi-class metrics).

Our state, a protein sequence, does not make a very good input into most standard machine-learning algorithms. These sequences vary in length and because of insertions and deletions exhibit some positional dependence. Here, our task is to map the sequences to a unique set of fixed length features and attempt to remove this positional dependence. Given that the protein sequences consist of a fixed alphabet of residues, our first attempt was to look at the relative frequencies of each residue in a protein sequence. However, as described in previous literature, this is not a very expressive descriptor. Indeed, for the 27 fold test case published before, the discriminative power of amino acid composition is around 50%. The performance in our blind 53-fold experiment is about 20% (5.3). This gives a baseline to understand the difficulty one the new, harder dataset.

The results in 5.1 average the fold level results over the SCOP class categories. The results are broken down by feature type: composition, secondary structure topology and both. These results show that proteins made primarily of alpha helices, SCOP classes a, f and g. This discrepancy between alpha helix and sheet based folds seems to follow the

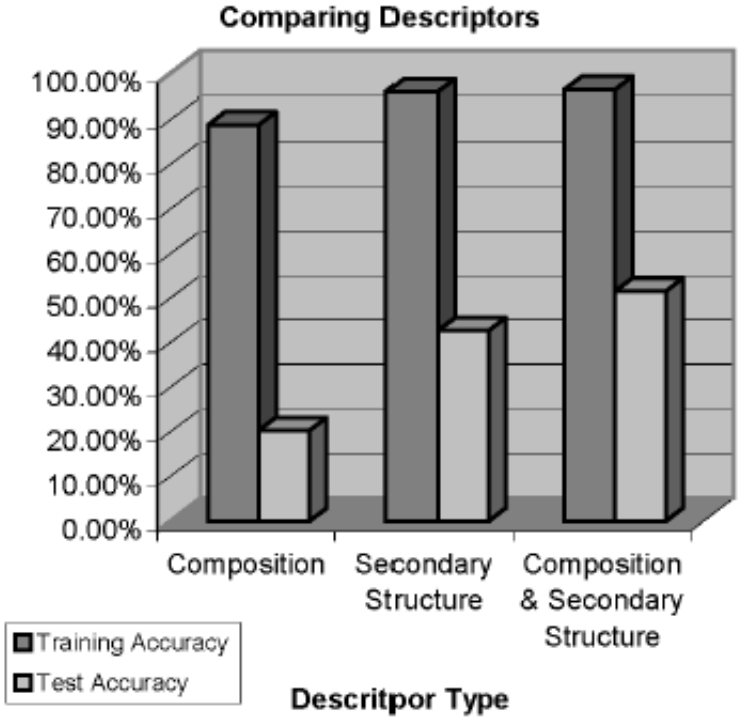


Figure 5.3: Overall accuracy of the various descriptors tested on the new dataset.

Table 5.1: SCOP Fold-level prediction results

| | Composition[a] | | SST[b] | | Composition and SST[b] | |
|---|---|---|---|---|---|---|
| Class | Accuracy | Confidence | Accuracy | Confidence | Accuracy | Confidence |
| a | 29.09 | 39.73 | 49.09 | 54.33 | 58.18 | 39.42 |
| b | 22.00 | 22.44 | 33.33 | 29.19 | 38.67 | 39.42 |
| c | 15.00 | 23.65 | 49.29 | 47.50 | 52.86 | 54.11 |
| d | 13.33 | 27.81 | 37.78 | 56.45 | 38.89 | 50.15 |
| f | 30.00 | 75.00 | 40.00 | 44.44 | 70.00 | 87.50 |
| g | 63.33 | 77.78 | 33.33 | 43.21 | 63.33 | 82.14 |
| Average | 22.64 | 31.95 | 41.70 | 44.96 | 48.49 | 53.74 |

[a] Composition: amino acid composition.

[b] SST: new secondary structure topology descriptor.

inherent higher error of beta sheet prediction except that this protocol also does well on beta-alpha-beta folds. Thus, the deficiency could also arise from the descriptor, which can

better encode beta-alpha-beta than beta or alpha folds or even beta and alpha (segregated) folds. Nevertheless, the secondary structure topology descriptor discriminates much better than composition alone.

To evaluate our SVM setup, we compared the current performance with a threading protocol PROSPECT Xu and Xu (2000). The program is downloaded from Ying Xu's webpage at the University of Georgia. We ran the threading program on each of our test sequences using a fold library comprised of our training set. The program was not manually tuned, instead uses the default parameters. In the end, we calculated the accuracy in the same manner as the analysis of the SVM performance. The overall accuracy from threading is 56.2%, which is higher than the 48.5% from SVM. The performance on individual classes is: 72%, 52% 52%, 47%, 30%, and 77%. Notice that threading utilized more information from the sequences and structures than SVM, such as structural contacts, statistical potentials, and sequence similarities; it is expected SVM performance will increase when such information is included. On the other hand, we want to emphasize that the current threading setup does not reflect the true performance of state-of-the-art in threading, which requires a larger template structure database and human intervention. This comparison between SVM and threading is for the purpose of providing a quick evaluation of SVM to ensure that the current performance is reasonable.

### 5.1.3 Discussion

We have found carefully designed secondary structure descriptors perform much better than amino composition and previously described features. In the 53 fold dataset, the composition alone can generate only 20% accuracy due to the larger number of folds. Our secondary structure descriptors improved the accuracy to 40%, a significant improvement when compared with mechanically designed descriptors reported to have worse performance than composition Dubchak *et al.* (1999). Finally, we compared our results directly to the current state-of-the-art in fold recognition, threading, and found that our results are comparable given limited information utilized by our protocol.

## 5.2 Disease *ns*SNP Identification

A single nucleotide polymorphism or SNP results from the mutation of a single nucleotide base pair in the genome. These SNPs account for much of the phenotypical variation among humans and, every so often, such mutations result in disease. The disrupted mechanism resulting in disease depends on the location of the mutation *i.e.* a mutation in the coding region *may* cause an amino acid substitution in the protein product; this is known as a non-synonymous SNP or as a single amino acid polymorphism (SAP). Even with the relative rarity of coding regions in the genome, SAPs account for roughly 50% of the genetic diseases caused by point mutations. A number of large scale high throughput

experiments have accumulated considerable SAP related data but fail to characterize SAPs in terms of disease and the underlying causes are poorly understood.

### 5.2.1 Methods

The SAP examples were collected from the variant pages of the Swiss-Prot knowledgeable version 49 Yip *et al.* (2004). The structural data for each corresponding SAP was collected from the ModSNP database Yip *et al.* (2004) which contains homology models for both wild type and variant proteins. After filtering problematic homology models from the dataset, 3438 SAPS located on 522 proteins were left; of these, 2249 had the disease phenotype and 1189 were tolerated.

The feature representation of each example SAP consists of both previous and new features. Here, the more interesting attributes are covered (the rest can be found in Ye *et al.* (2007)). First, we use a variant of the SIFT score Ng and Henikoff (2001) that considers both the raw score as well as the difference between the wild type and variant residue. Another feature encodes the solvent accessibility of the residue Ye *et al.* (2007); Dobson *et al.* (2006). Here, we investigate a slightly different definition that uses the $C_\beta$ rather than the full side-chain and thusly does not require a full model; this gives it a wider applicability. Next, we consider any nearby functional sites as well as the full structure model energy; if a SAP occurs on or near a functional site or destabilizes the protein hydrophobic core, then it could cause disease. We also consider disulfide or hydrogen bonding patterns around the SAP site as well as its occurrence in a disordered region. Finally, we consider the aggregation properties of the protein as calculated by TANGO Fernandez-Escamilla *et al.* (2004). Since some diseases are associated with protein aggregation, it makes since to check if a SAP could cause such aggregation.

### 5.2.2 Results

This study starts by investigating the predictive power of individual attributes Ye *et al.* (2007). We found that SAP frequency, conservation and solvent accessibility had the greatest predictive power alone; this is consistent with previous studies Bao and Cui (2005); Dobson *et al.* (2006). Nevertheless, our study found two new attributes that achieved higher alone predictive power than solvent accessibility: structural neighbors and nearby functional sites. These results can be seen in 5.2.

The structural neighbors comprise a microenvironment around the SAP. Certain microenvironments tolerate SAPs while others do not; this coincides with previous observations Wei and Altman (2003). Moreover, a SAP in the vicinity of a functional site often leads to disease. That is, a SAP near a ACT_SITE, BINDING, METAL, MOD_RES and DISULFID have a higher ratio of disease to polymorphism. Likewise, SAPs in a transmembrane region overwhelmingly lead to disease.

Table 5.2: Performance of specific attribute(s) subsets

| Attribute Groups | Accuracy | Correlation |
|---|---|---|
| Residue Frequencies | 77.5 | 0.489 |
| Structure Neighbor Profile (13Å) | 76.4 | 0.467 |
| Conservation Scores | 74.8 | 0.425 |
| Nearby Functional Sites | 69.3 | 0.246 |
| Solvent Accessibility | 68.0 | 0.232 |
| $C_\beta$ Density | 67.0 | 0.190 |
| Final attribute set | 82.6 | 0.604 |

The final model achieved an accuracy of 82.6% and a correlation coefficient of 0.604 using 5-fold cross-validation. This performance improves on all previous work Dobson *et al.* (2006); Bao and Cui (2005) by a significant amount. It improves on the SIFT Ng and Henikoff (2001) score by 0.124. The next best correlation coefficient was 0.49 Dobson *et al.* (2006), an improvement of 0.065 over SIFT.

### 5.2.3 Discussion

This work provides further evidence for the biochemical mechanisms that lead to disease causing SAPs. It has overcome limitations of previous work and investigated a number of new attributes. Moreover, the predictive power of our protocol far exceeds previous work. To this end, we have created a freely available web server called SAPRED at http://sapred.cbi.pku.edu.cn/. This server takes a FASTA format protein sequence and mutation; it then determines its structure and predicts if the mutation could cause disease.

## 5.3 Membrane-binding Annotation

A large number of cytosolic proteins bind reversibly to the membrane in order to perform their function. This reduces a three-dimensional search for a binding partner to a restricted two-dimensional search; this search is restricted even further to specific lipids following specific binding interactions. These proteins can be identified in painstaking experimental and translocation studies Mozsolits and Aguilar (2002); Wu and Brand (1994). Extending these experiments to the genome scale is not currently possible and traditional sequence analysis techniques cannot transfer sequence similarity to function. Moreover, structural similarity can only transfer function roughly two-thirds of the time. In our work Bhardwaj *et al.* (2006), we developed an automated prediction protocol that utilizes

structure-based and sequence-based features to identify peripheral membrane binding proteins. In more recent work, we explore the performance of a number of classifiers over this same dataset and features; while SVM is a robust classifiers, it is not known beforehand which classifier works best. Moreover, we analyze the features important to membrane-binding in the context of the model.

### 5.3.1 Methods

The classification dataset is constructed as follows: the RCSB PDB database is searched for all modular membrane-targeting (see 5.4) domains resulting in 146 proteins. Subsequently, the redundancy is reduced such that no two proteins have more than 40% sequence identity yielding 40 proteins. A representative protein set of 250 proteins is taken from previous work Stawiski *et al.* (2003) and filtered for any lipid binding proteins giving 230 proteins as a negative set. Each example protein structure is encoded using a set of numerical attributes similar to those proposed in our previous work Bhardwaj *et al.* (2005a).

The membrane-binding identification experiment is setup as a standard classification problem. This problem can be handled by classifiers available in the *malibu* workbench such as support vector machines, boosted trees, boosted C4.5, boosted stumps and C4.5. Each of these algorithms has been introduced in Chapter 2 and Chapter 4. Given the small size of the dataset, leave-one-out cross-validation (often confused with jackknife) is used to evaluate the accuracy of the model.

### 5.3.2 Results

The membrane inside the cell composes a number of lipids with varying charge. The most negative of these are phosphoinositides, which function as recruiters in local anionic clusters McLaughlin *et al.* (2002). Therefore, electrostatics play a large role in protein membrane interactions and accordingly we incorporate net charge and electrostatic patch (see Bhardwaj *et al.* (2006) for details) as features. To validate the use of these cationic patches used in the prediction, we observed their position relative to the membrane. In 5.5, five proteins are shown where two Malmberg *et al.* (2003); Kohout *et al.* (2003) have experimentally determined orientations and three have proposed orientations Stahelin *et al.* (2003); Lemmon *et al.* (2002). On each protein the four largest cationic patches are colored blue, cyan, white and pink, respectively. Indeed, in four of the five proteins at least two of patches interact with membrane and in only one case the largest patch is in contact. While these cationic patches are good indicators for membrane protein prediction, they are not conclusive.

Since this is the first such machine learning method applied to membrane-binding proteins, there is no dataset to compare. 5.3 compares the results from two publications

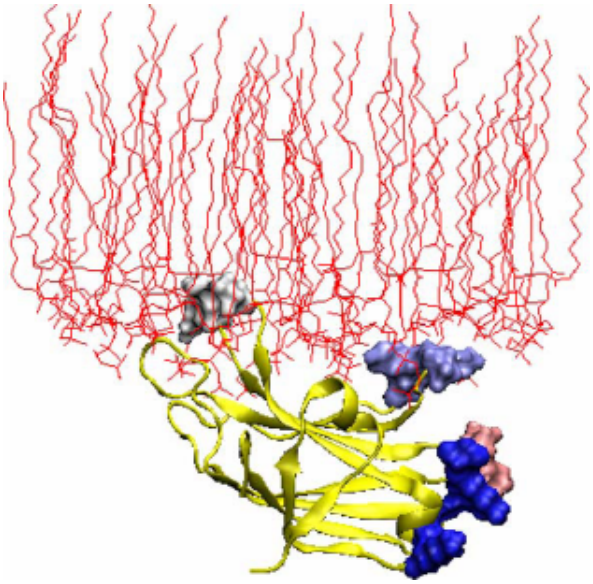

Figure 5.4: An example membrane-binding protein with the relative orientation of binding. This figure shows the C2 domain of PKC-$\alpha$ (1DSY), which is involved in $Ca^{2+}$ - dependent membrane signaling. The protein is shown in a yellow cartoon representation where the first four largest cationic patches have been mapped on the surface and are colored according to their order on a blue-white-red scale: largest patch in blue, second largest in light blue and third and fourth largest in white and light red, respectively. The membrane without hydrogens is shown in red.

Bhardwaj *et al.* (2006); Langlois *et al.* (2007). The support vector machine (SVM) classifier in the first work (JMB06) Bhardwaj *et al.* (2006) was trained with extensive model selection achieving 93.7% accuracy and the most balanced sensitivity and specificity. This result was achieved by tuning the cost parameter for class labels in SVM.

This work in Langlois *et al.* (2007) compares a set of classifiers over the same dataset without balancing the sensitivity and specificity. Instead, the area under the ROC curve is used to estimate how the classifier will perform over other distributions. In 5.3, all the classifiers do well in terms of accuracy comparing boosted trees (best) at 93.4% accuracy with a single decision tree (the worst) at 88.6%. Nevertheless, the area under the ROC gives more meaningful results. That is, the boosted tree classifiers perform best, followed by SVM, then boosted stump and finally the C4.5 decision tree. The area under the ROC measures the number of (bubble) sorts it would take to separate a set of predictions based on class value. Thus, a highly confidence yet incorrect value will decrease the area under the ROC more than a less confident incorrect value; in both cases, the accuracy would remain the same. Thus, with 93% area under the ROC, confidence predictions by boosted trees will be far more accurate than those by C4.5 with 65.5% area under the ROC.

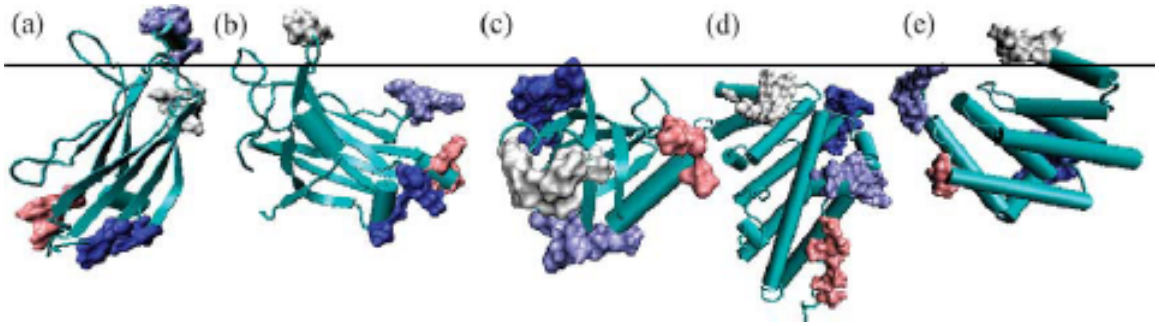


Figure 5.5: The relative orientation of cationic patches with respect to membrane-binding surfaces. Membrane-binding surfaces were determined experimentally for (a) the cytosolic phospholipase $A_2$ C2 domain Malmberg *et al.* (2003) and (b) the PKC$\alpha$ C2 domain Kohout *et al.* (2003). Membrane-binding surfaces have been proposed, on the basis of biophysical and mutational studies, for (c) the $\beta$ - spectrin PH domain Lemmon *et al.* (2002), (d) the CALM-ANTH domain Stahelin *et al.* (2003), and (e) the epsin 1-ENTH domain Stahelin *et al.* (2003). Proteins are shown in cartoon representations and cationic patches are displayed using the "surface" representation. The RGB scale is used to color the patches with blue, cyan, white, and pink representing the largest, the second largest, third largest, and the fourth largest patches, respectively. A continuous line indicates the location of the lipid head group region in the lipid bilayer. The figure was made with VMD Humphrey *et al.* (1996).

Table 5.3: Comparing classifiers over the protein-membrane set using leave-one-out cross-validation

| Publication | Algorithm | Accuracy | Sensitivity | Specificity | Area Under ROC |
|---|---|---|---|---|---|
| JMB06 | SVM | 93.7 | 84.8 | 95.0 | - |
| ABME06 | Boosted Trees | 93.4 | 67.5 | 97.9 | 93.4 |
| | Boosted C4.5 | 92.3 | 57.5 | 98.7 | 93.6 |
| | Boosted Stumps | 91.6 | 65.0 | 96.1 | 84.6 |
| | SVM | 86.8 | 65.0 | 90.6 | 88.1 |
| | C4.5 | 88.6 | 55.0 | 94.4 | 65.5 |

Next, we analyze a truncated boosted stump model for insight into feature importance in a machine learning model. 5.6 illustrates the model learned by boosting one-level decision trees (decision stumps) over the protein-membrane dataset. The root of each decision stump contains a decision: if true the left leaf is used otherwise the right leaf is used. Every leaf gives a measure of confidence in its prediction and serves as a weight for the

AdaBoost algorithm. The final decision is reached by summing over all the chosen leaves: if the total is greater than zero the protein is predicted as binding membrane otherwise not. This model in 5.6 was stopped after 13 iterations to give a more interpretable model. Indeed, the leave-one-out cross-validation accuracy of the model stopped at 13 iterations reaches 90% (91.6%) over the protein-membrane dataset. Thus, this model contains a majority of the information used in the extended model of 1024 iterations.

One observation that can be made over this model is the predominance of sequence-based features over the structure-based. That is, the size of the largest surface patch does not play a significant role in discriminating membrane-binding proteins. Another observation entails the duplication of features. In this model charge appears twice; this serves to illustrate one method the AdaBoost algorithm employs to achieve good generalization. Specifically, it widens the $l_2$ margin Schapire and Singer (1999) by refining and expanding the rules learned.

Several interesting rules can be extracted from 5.6. For example, if a protein has more than 5.3% surface valine or more than 15% surface serine, it is more likely to bind membrane. Both of these are small, neutral amino acids important to binding. Since the protein does not immerse itself in the membrane, we do not expect a significant number of surface hydrophobic residues; in the model we do not find any. Likewise, in this model, we see a number of exclusionary rules. Such rules do not directly give any information about the membrane-binding proteins but do perform a useful function by weeding out proteins that are far away (from membrane-binding) in terms of some characteristic. Specifically, both models exclude proteins with a larger proportion of glycine, *i.e.* proteins that are overly flexible in some way.

### 5.3.3 Discussion

This is the first such method developed to identify membrane-binding proteins achieving reasonable accuracy given the limited data. Specifically, a good portion of this accuracy can be attributed to a biologically motivated electrostatic patch descriptor. This work not only investigates the efficacy of using machine learning to identify membrane-binding proteins, it culminates in a resource that provides online prediction for membrane-binding proteins. One future direction will be to combine this technique with homology modelling or other fold recognition techniques (as introduced in the first section) to perform a genome scale prediction of membrane-binding proteins. Such a scan will serve as initial screening for potential experimental candidates.

A number of factors are left unaccounted for by this protocol. For example, it has been shown that calcium binding and phosphorylation (dephosphorylation) greatly enhance membrane affinity Cho and Stahelin (2005). Likewise, the presence of intrinsically unstructured proteins whose membrane-binding surfaces may be induced by contact with membrane McLaughlin and Murray (2005) should also be taken into account. Our current protocol applies to modular, unaltered structural domains since these are the only ones

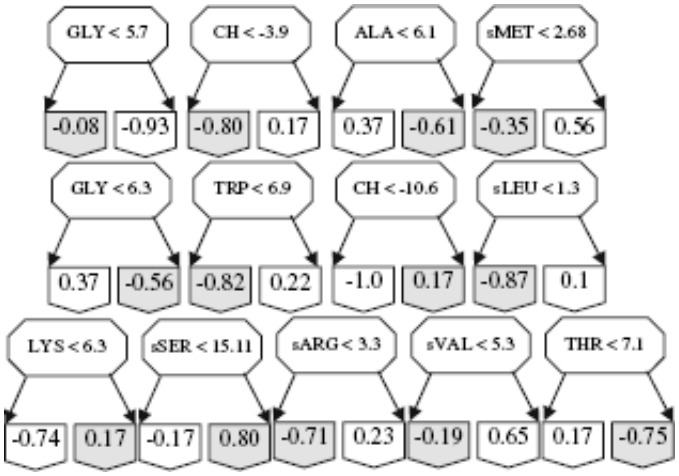


Figure 5.6: A graphical representation of the boosted stump model built on the protein-membrane dataset. At the root of each node, a single decision is made testing whether the feature in question is less than a learned threshold; if so, the value on the left leaf is used, otherwise the right leaf is used. The final decision is made by summing up these values; if the final value is greater than zero, the protein is predicted to bind membrane otherwise it is predicted not to bind membrane. The s in sASN stands for surface amino acid composition (of asparagine).

that could be collected for our dataset. However, we also plan reformulate the current problem to extend this protocol to sequence-based data.

## 5.4 Summary and Future Work

This work illustrates machine learning problem formulations in biology. Each problem is best formulated in the classification setting; however, each problem has its own particular features that must be carefully considered in order for the results to have meaning. For example, a large portion of protein sequence data is redundant and related; this relation/redundancy can unfairly bias both training and testing of a machine learning algorithm. One method to handle this potential bias is to reduce the sequence identity of the dataset. It may be argued that this is insufficient to remove all possible bias but no other method is currently available. Likewise, when performing cross-validation there are cases where examples must be appropriately grouped. That is, in the disease SAP prediction problem, several SAPs may belong to the same protein; thus, these SAPs should be grouped into the same partition.

While these works appear unconnected, each fulfills one step in a framework with the ultimate aim of understanding the underlying mechanisms that govern genetic disease.

Specifically, one cause of genetic disease is a point mutation in a coding region of DNA leading to a single amino acid polymorphism or SAP. Accurately predicting disease causing SAPs requires the assignment of structure and fold to sequence when sequence homology fails. Thusly, given a sequence and mutation our protocol assigns structure either using homology modeling techniques or failing that our proposed fold recognition protocol. Subsequently, our technique for identifying disease-causing SAPs is used to predict if the proposed mutation causes disease. Finally, our function annotation method identifying membrane-binding proteins can be generalized to other functions (see Chapter 6) and give insights into the wider functional effects of a mutated protein.

There are a number of ways this research can be extended. First, we need to generalize the protein function annotation protocol to handel functions other than membrane-binding. Second, we need to identify potential processes that a subject protein may be involved. Note, the process is the proteins global contribution whereas its function is the mechanism by which it contributes. Finally, one of the important uses of this system is that it can be perturbed to create hypothetical diseases. These diseases can then be matched to real cases and thusly the underlying disease mechanisms can be divined.

# Chapter 6

# Annotation of DNA-binding Proteins

In the previous chapter, a framework is introduced to identify underlying mechanisms of genetic disease. One important step in this framework is the identification of protein function (in the previous chapter this focused on membrane-binding proteins). In this chapter, we cover another application of function prediction, the identification of DNA-binding proteins and explore function annotation on several levels. First, we introduce a method that identifies DNA-binding proteins using structure; this method outperformed all previously published works. Next, we generalize this method to the sequence level; this allows the classifier to leverage the larger amount of available sequence information. Finally, we cover a method for determining the DNA-binding residues on a protein; this method has also proven competitive to previously published work.

## 6.1 Background

DNA-binding proteins themselves are fundamentally important, found at the heart of cellular regulation and chromosome maintenance Luscombe *et al.* (2000). These proteins comprise roughly 7% of proteins encoded in the eukaryotic genome and 6% in the prokaryotic genome Luscombe *et al.* (2000). They also represent a diverse set sequences, structures and functions. Indeed, Luscombe and Thornton (2002) classified DNA-binding proteins into 54 structural-families. Thus, conventional sequence similarity methods may not lead to reliable annotations.

Structural genomics projects promise to provide detailed three-dimensional coordinates for every protein fold. These projects target proteins with little known information aiming to identify those with novel folds. However, a protein fold does not uniquely define a single function. Identifying the correct function will come by analyzing both sequence and structural properties including active sites, charge, etc. In addition, many of these methods produce low-resolution structures that show only the global fold. Thus, it is favorable to have an efficient method that can utilize limited information even if its just

the boundaries of a structural domain.

Indeed, robust experimental techniques that identify DNA-binding proteins include domain directed filter binding assays Cajone *et al.* (1989), chromatin immunoprecipitation on microarrays (ChIP-chip) Buck and Lieb (2004); Ren *et al.* (2000), genetic analysis Freeman *et al.* (1995) and x-ray crystallography Chou *et al.* (2003). Similar techniques available to identify binding residues involve directed mutagenesis studies Ruvkun and Ausubel (1981). Such experiments are very time consuming and labor intensive especially to study all possible residues or proteins. Automated recognition of binding proteins and their residues can narrow the scope of such searches saving considerable time and labor.

Given the importance of automatic prediction methods, a number of *in silico* efforts have investigated informative feature representations of both DNA-binding proteins and their binding residues over a range of machine learning algorithms. For example, several groups have investigated structure-based features Jones *et al.* (1999) for specific DNA-binding structural motifs (*e.g.* helix-turn-helix) achieving 78% and 71% accuracy using a hidden Markov model Shanahan *et al.* (2004); Pellegrini-Calace and Thornton (2005). Likewise, previous work has also focused on a wider range of DNA-binding proteins using a more general set of structure-based attributes in conjunction with neural networks Ahmad *et al.* (2004); Ahmad and Sarai (2004) achieving 83% accuracy. This trend was further extended to low resolution homology models using logistic regression and a novel set of structure descriptors Szilagyi and Skolnick (2006).

Identifying specific protein-DNA interaction sites composes two symmetrical problems: finding where the protein binds on the DNA and where the DNA binds on the protein. The first problem is considerably harder given the greater flexibility and limited code of DNA translates to a wider variety of binding methods. However, this problem has been addressed with much attention. Specifically, a number methods have been developed including information content Schneider *et al.* (1986), consensus sequence Day and McMorris (1992), weight matrices Stormo *et al.* (1982) and protein-DNA docking Sarai and Kono (2005). Nonetheless, few studies have addressed the problem of identifying the DNA-binding site on the protein. Specifically, Ahmad *et al.* Ahmad *et al.* (2004) used a neural network classifier trained on sequence features to discriminate DNA-binding residues from non-binding on a set of 62 DNA-binding proteins; using 3-fold cross-validation they achieved 40% sensitivity and 76% specificity. They improved these scores by adding structure features achieving 40% and 81.8% Ahmad *et al.* (2004) and sequence conservation 68% and 66% Ahmad and Sarai (2005). Later work by Kuznetsov *et al.* Kuznetsov *et al.* (2006) used support vector machines (SVM) trained on both sequence and structural attributes to achieve 71% accuracy with a balanced sensitivity and specificity; by adding evolutionary conservation the sensitivity and specificity were improved to 79% and 85%. Further sequence-based prediction by Yan *et al.* Yan *et al.* (2006) achieved 71% (leave-one-protein-out) accuracy using a Naïve Bayes classifier over 171 proteins. Finally, Bhardwaj and Lu Bhardwaj and Lu (2007) addressed two primary

concerns with previous studies: bias involved with using every residue as opposed to just the surface residues and the rather small test sets. Under these more stringent conditions, they trained an SVM classifier on sequence and structural features and performed subsequent post-processing techniques to achieve a sensitivity and specificity of 71% and 69%, respectively.

## 6.2 Protein-DNA Identification: Structure

This section covers the identification of proteins that bind DNA, which can be naturally posed as a classification problem. First, we describe a machine learning protocol that accurately predicts DNA-binding proteins given a crystal structure Bhardwaj *et al.* (2005a). Second, we address what is the best classifier for this domain and what features contribute in the final predictive model Langlois *et al.* (2007).

### 6.2.1 Methods

Two datasets were used in this work. The first dataset consists of 121 proteins that bind DNA and 238 that do not; this set was gathered from previously published works Stawiski *et al.* (2003); Ahmad *et al.* (2004); Jones *et al.* (1999). Each protein has a resolution better than 3 Åand no two proteins share more than 35% sequence identity. The second dataset is a subset of this one with no two protein sharing more than 20% sequence identity; this yielded 75 DNA-binding proteins and 214 non-DNA-binding proteins.

In order to apply a classifier to a set of protein data, each protein example must be encoded into a vector of numerical characteristics often called attributes or features. These features comprise two general types: sequence-based and structure-based features. The sequence-based features include net charge as assigned by the CHARMM force field parameters Brooks *et al.* (1983) and amino acid composition. The structure features include electrostatic patch size and surface amino acid composition. A residue is considered a surface residue if more than 40% of its surface area is accessible to solvent.

The electrostatic patch was grown as follows: the Delphi Gilson *et al.* (1988) program was used to calculate all electrostatic interactions and the CHARMM force field was used to assign partial charges. The positive surface patches were found using an iterative growing algorithm where a surface atom (using the previous definition) with more than 200 kT/e was used as a starting point. The algorithm, then, iteratively adds all positive atoms within two angstroms. After all the patches have been identified, the patches are sorted according to size and the size of the largest patch forms the final feature. 6.1 illustrates the correlation of positive surface patches with the location of the DNA binding site.

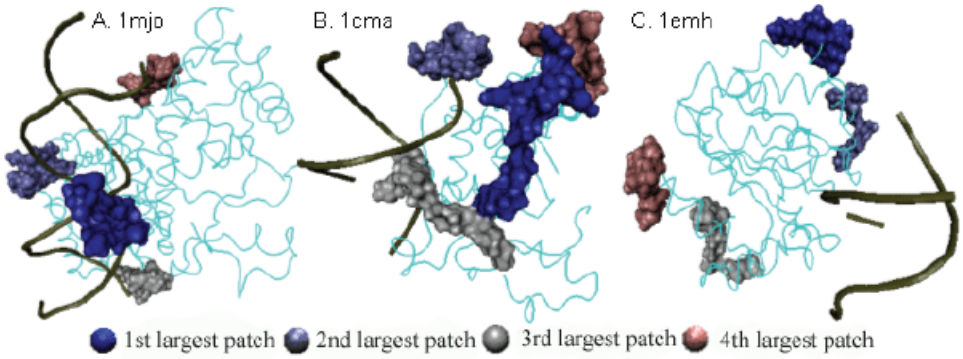


Figure 6.1: Orientation of the four largest surface patches with respect to the DNA. The protein is shown in 'tube' representation colored in cyan. The DNA is displayed in a golden color with only the backbone shown. Patches were mapped on the surface by adapting the PDB files for atom labeling in each patch and visualized in VMD Humphrey *et al.* (1996).

### 6.2.2 Results

The results in 6.1 must be considered separately in two categories: training and testing performance. The training performance (with the exception of self-consistency) is the performance used to select parameters and serves as a useful measure to compare the polynomial kernel to the linear kernel and feature subsets. As shown in 6.1, there is a slight improvement when the patch size is added to overall charge in the linear kernel, 82% to 83% accuracy. Moving to the polynomial kernel and all the features, the training performance improves significantly from 83% to 90%. Note, the self-consistency demonstrates that the polynomial kernel is expressive enough to fit all the data but without any left out test data there is no way to know whether this model overfits the data.

The testing results (suffixed with HO) (in 6.1 demonstrate that the polynomial SVM performs nearly as well on the test data. That is, when half the data is used for training (5-fold cross-validation on this half to select parameters) then the held out portion is tested on, this classifier achieves 83% accuracy with balance sensitivity and specificity. Using a method that repeatedly samples one positive and one negative for testing (1P-HO or one-pair holdout), this classifier achieves a competitive 86% accuracy with a fairly balance sensitivity and specificity. Reducing the dataset such that no two proteins share more than 20% identity, reduces the accuracy of this method to only 85%.

6.1 exemplifies the performance of our SVM classifier in three cases. The first case, 1MJO, was both correctly classified and its largest positive patch overlaps the DNA-binding residues. The second case, 1CMA, was correctly predicted as DNA-binding yet the largest patch does not overlap the DNA-binding residues. And finally, the third case, 1EMH, was predicted incorrectly; none of the patches overlap with the DNA-binding site.

Table 6.1: Performance of SVM on the Protein-DNA dataset

| Descriptors | Kernel | Validation | Accuracy | Sensitivity | Specificity | Parameters | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | D | C | $\gamma$ |
| Charge | Linear | LOOCV[b] | 82.4 | 58.6 | 95.7 | 1 | 11 | 0.01 |
| Charge+PS[a] | Linear | LOOCV | 83.8 | 60.3 | 96.6 | 1 | 13 | 0.02 |
| All | Polynomial | Self[c] | 100 | 100 | 100 | 2 | 10 | 0.309 |
| | | LOOCV | 90.5 | 81.8 | 94.9 | 2 | 19 | 0.054 |
| | | 5-fold CV[d] | 89.1 | 82.1 | 93.9 | 2 | 19 | 0.051 |
| | | 5-fold CV(20%) | 90.3 | 67.4 | 94.9 | 2 | 23 | 0.034 |
| | Polynomial | HO[e] | 83.3 | 82.5 | 83.5 | - | - | - |
| | | 1P-HO[f] | 86.3 | 80.6 | 87.5 | - | - | - |
| | | 1P-HO(20%)[g] | 85.8 | 81.6 | 87.8 | - | - | - |

[a] PS: size of largest patch.

[b] LOOCV: leave-one-out cross-validation.

[c] Self: self-consistency.

[d] 5-fold CV: 5-fold cross-validation.

[e] HO: holdout validation.

[f] 1P-HO: iterated leave pair out.

[g] 20%: dataset reduced to 20% sequence identity.

This serves to illustrate that electrostatics alone do not govern residue binding yet they do make a good indicator.

In previously covered work Bhardwaj *et al.* (2005a), the classifier used was support vector machines (SVM). In subsequent work Langlois *et al.* (2007), we analyzed the performance over a number of learning algorithms. Unlike SVM, many classifiers do not support cost-sensitive learning; thus, for a fair comparison, each of the classifiers minimizes the classification error. This accounts for the more unbalanced results in 6.2. In this table, the boosted tree classifiers perform best in terms of every metric; given the simpler model selection, the poorer results with SVM could mean the range of parameters was too narrow. Another observation from 6.2 follows the similarity in boosted C4.5 and boosted trees, our simpler decision tree algorithm. This suggests that a more robust decision algorithm does not matter in boosting. Finally, the boosted stump classifier performs nearly as well as the SVM classifier and boosted trees. This makes the model of the boosted stump algorithm a good approximation of the features important in these other more powerful algorithms.

The results in 6.2 illustrate that the boosted stump model over the protein-DNA dataset. This model was stopped early at 13 iterations to give a more interpretable model achieving 83% accuracy compared to 85% for the full model. Thus, this model contains a majority of information used in the final boosted stump model. Several observations

Table 6.2: Comparing classification and evaluation methods over the protein-DNA dataset

| Validation | Algorithm | Accuracy | Sensitivity | Specificity | Area Under ROC |
|---|---|---|---|---|---|
| 2-CV[a] | Boosted Trees | 86.5 | 61.4 | 95.3 | 88.0 |
| | Boosted C4.5 | 86.3 | 61.5 | 95.0 | 88.8 |
| | Boosted Stumps | 83.6 | 60.5 | 91.7 | 81.6 |
| | SVM | 84.5 | 57.5 | 94.0 | 84.4 |
| | C4.5 | 79.4 | 59.8 | 86.3 | 61.3 |
| 5-CV | Boosted Trees | 87.2 | 63.8 | 95.4 | 88.4 |
| | Boosted C4.5 | 86.6 | 61.4 | 95.4 | 89.6 |
| | Boosted Stumps | 84.1 | 61.2 | 92.2 | 82.7 |
| | SVM | 85.7 | 59.1 | 95.0 | 85.9 |
| | C4.5 | 79.4 | 59.9 | 86.3 | 54.1 |
| LOOCV[b] | Boosted Trees | 88.5 | 66.7 | 96.3 | 88.7 |
| | Boosted C4.5 | 86.5 | 61.3 | 95.3 | 89.8 |
| | Boosted Stumps | 85.1 | 62.7 | 93.0 | 84.6 |
| | SVM | 86.3 | 62.7 | 93.9 | 86.3 |
| | C4.5 | 80.8 | 65.0 | 85.0 | 74.0 |

[a] CV: cross-validation

[b] LOOCV: leave-one-out cross-validation

can be made from this model. First, a good number of features represent sequence and to a lesser degree structure. Second, a majority of features comprise known DNA-binding residues including arginine (internal and surface) and those found on the largest electrostatic patch.

### 6.2.3 Discussion

In these works, we investigate the ability of a classification model to identify DNA-binding proteins based on a crystal structure. Further, we compared a number of algorithms and found boosted trees work best achieving 88% accuracy. Using a less complex version of boosted trees called boost stumps (single node decision trees), we investigated the features important in the context of the model. Indeed, we found the size of the largest patch, charge and arginine content to be important features; consistent with expectations about DNA-binding proteins. Comparing the various algorithms, boosting improves 8% over a single C4.5 tree. Both boosted stumps and SVM fall in between given their simpler models and more complicated model selection types.

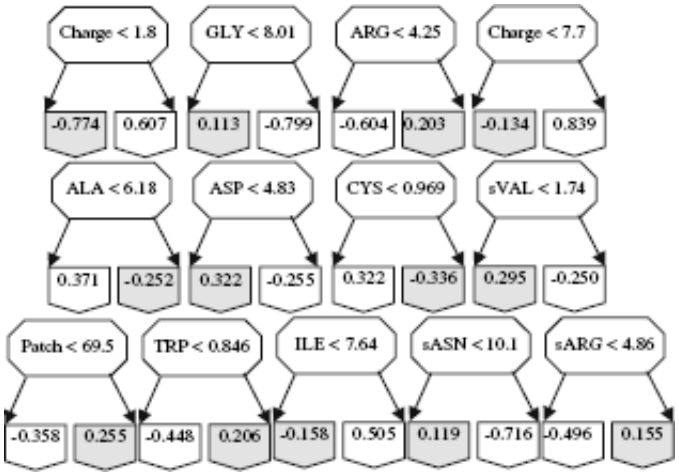


Figure 6.2: A graphical representation of the boosted stump model built on the protein-DNA dataset. At the root of each node, a single decision is made by testing whether the feature in question is less than a learned threshold; if so, the value on the left leaf is used, otherwise the right leaf is used. The final decision is made by summing up these values; if the final value is greater than zero, the protein is predicted to bind DNA otherwise it is predicted not to bind DNA. The s in sASN stands for surface amino acid composition.

## 6.3 Protein-DNA Identification: Sequence

In many cases, only the sequence and structural domain of a protein is known. To this end, we have developed an accurate sequence-based protocol to identify DNA-binding proteins given a sequence domain. The closest published work Szilagyi and Skolnick (2006) investigated the applicability of building homology models and calculating structure-based features. In this work, we compare our new sequence-based prediction protocol directly to all previous work and show that it works competitively to most structure-based methods using only sequence information.

### 6.3.1 Methods

In order to compare with previously published work, we have recreated the five respective published datasets (see 6.3). The DNA-binding proteins for each dataset originated from one of two sources: the work of Luscombe *et al.* Luscombe and Thornton (2002) and the Nucleic Acid Database Berman *et al.* (1992). Firstly, based on Luscombe *et al.* Luscombe and Thornton (2002), Stawiski *et al.* Stawiski *et al.* (2003) collected a set of 54 DNA-binding protein chains with no more than 35% sequence identity. Subsequently, Ahmad *et al.* Ahmad and Sarai (2004) compiled a set of 78 DNA-binding protein chains from a set of 62 representative structures and Bhardwaj *et al.* Bhardwaj *et al.* (2005a) combined

Table 6.3: Statistics of the five datasets used in this work

| | Example | Pos. | Neg. | Identity |
|---|---|---|---|---|
| JMB03 | 304 | 54 | 250 | 35/25 |
| JMB04 | 188 | 78[a] | 110 | ?/25 |
| NAR05 | 359 | 121 | 238[a] | 35/? |
| JMB06 | 248 | 138 | 110 | 35/25 |
| ABME06 | 289 | 75[b] | 214 | 20 |
| NEW07a | 388 | 138 | 250 | 35/25 |
| NEW07b | 372 | 122 | 250 | 25/25 |

[a] Sequences with greater than 90% identity exist in dataset.

[b] Single sequence misclassified.

the datasets of both Stawiski and Ahmad producing a set of 121 DNA-binding protein chains. Langlois *et al.* Langlois *et al.* (2007) filtered the Bhardwaj dataset to 20% identity yielding 75 protein chains. Secondly, Szilagyi *et al.* Szilagyi and Skolnick (2006) went on to construct the largest set of 138 DNA-binding protein chains using the Nucleic Acid Database.

Since a classifier requires both positive and negative examples, each of the aforementioned work also collected a set of proteins known not to bind DNA. Selected work Stawiski *et al.* (2003); Bhardwaj *et al.* (2005a); Langlois *et al.* (2007) used the PDBSELECT database Hobohm and Sander (1994) applying a 25% identity cutoff to yield 250 protein chains. Bhardwaj *et al.* Bhardwaj *et al.* (2005a) used a subset of the PDBSELECT, 238 proteins and Langlois *et al.* Langlois *et al.* (2007) used a more stringent 20% identity cutoff yielding 214 proteins from this same set. Other work Ahmad and Sarai (2004); Szilagyi and Skolnick (2006) started with a representative 126 set Rost and Sander (1993) excluding DNA-binding proteins giving a set of 110 protein chains.

We created two more datasets. The first dataset combines the positive set from Szilagyi *et al.* Szilagyi and Skolnick (2006) and the negative set from Stawiski *et al.* Stawiski *et al.* (2003) creating a dataset with 388 protein examples. The second dataset is similar to the first except that the proteins in the positive set have been filtered such that no two proteins share more than 25% sequence identity using the Pisces Server Wang and Dunbrack (2003).

Most supervised machine learning algorithms require the dataset to comprise a set of fixed length examples or feature vectors. In this work, we investigate a novel representation that encodes protein sequences called local environment amino acid composition. This feature representation defines a residue based on both its identity and its environment. One simple method to define the environment is to take some property, *e.g.*

hydrophobicity, and average it over a window around the amino acid of interest. Then we define some threshold, *e.g.* 0, and redefine the amino acid, *e.g.* Alanine, by both type and environment. That is, if the average hydrophobicity around a specific alanine exceeds 0, then it is counted as a hydrophillic Alanine otherwise it is counted as a hydrophobic Alanine. For just one such property, we have expanded the amino acid alphabet from 20 to 40 attribute types.

The current implementation utilizes the secondary structure propensities Prabhakaran (1990) using a window similar to the original Chou-Fasman Chou and Fasman (1978) of eight residues on either side (using any window size from 6-10 yields similar results). Three versions of this descriptor are calculated where for each version, two properties are considered. For example, in one version the helix and sheet propensity in the window around a specific amino acid is compared against a threshold of 1. Thus, for each amino acid, there are now four amino acid types: helix and sheet both exceed the threshold, helix exceeds the threshold, sheet exceeds the threshold and neither exceed the threshold. The other two versions consider (helix, turn) and (sheet, turn). Note, for each version we have 80 features yielding 240 when combining them together.

Two additional feature types compose the complete feature description. The first is dipeptide composition, which does not maintain the amino acid order (including unknown amino acids) comprising 231 features. The second feature type sums some property over the entire sequence. In this case, we considered only total charge over the protein. While normalizing these features to sequence length yields slightly better results, the normalized values are less interpretable in the ADTree model. In sum, there are 472 features to describe each protein.

This work focus on three main goals. Firstly, we demonstrate that our new features in combination with a classifier performs accurately compared to previous work. Secondly, we compare our sequence-based method to a standard sequence analysis technique, blast. Finally, we analyze the features in the context of the ADTree model in order to understand the characteristics define DNA-binding.

### Comparison to Previous Methods

The first goal of our work is to build a classifier that accurately predicts DNA-binding proteins using fast to calculate sequence characteristics alone. Since a majority of proteins do not have structural data, it would be desirable to build a pure sequence-based prediction protocol. With this view, we developed a sequence-based feature representation that encodes residue identity and local environmental characteristics.

We reconstructed five previous datasets for direct comparison Stawiski *et al.* (2003); Ahmad and Sarai (2004); Bhardwaj *et al.* (2005a); Szilagyi and Skolnick (2006); Langlois *et al.* (2007). Unlike previous work, we validate our protocol with 10-fold cross-validation as opposed to leave-one-out cross-validation (or a similar method). In addition, we constructed two new datasets combining the positive and negative sets from Szilagyi and

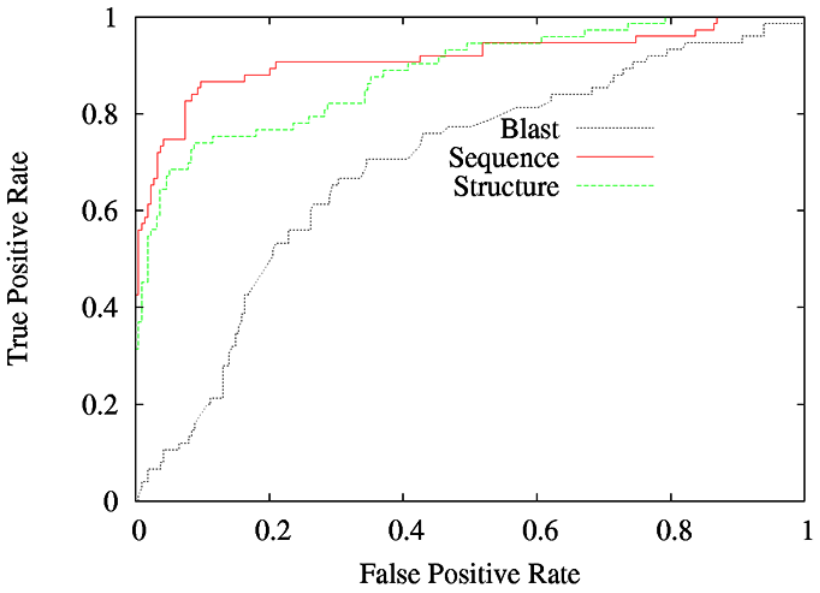


Figure 6.3: A ROC comparison of the new sequence-based feature representation with the ABME06 structure-based features using Boosted Trees and Blast.

Skolnick (2006) and Stawiski *et al.* (2003), respectively.

In general, the results using sequence-based characteristics are encouraging in 6.4. Specifically, the area under the ROC (AUR) on the JMB06 dataset is nearly the same for both our sequence-based and Szilagyi and Skolnick (2006) structure-based protocols. Moreover, the results on the ROC curve (6.4) show that neither of the curves dominate. Likewise, even with a dataset containing less homologous proteins (20% compared to 35%) our sequence-based protocol achieves 91% AUR outperforming the structure-based protocol with 88.7% AUR in the ABME06 paper Langlois *et al.* (2007); in the corresponding ROC curve (6.3) our sequence-based results dominate over a large range of class distributions. The one notable exception is the JMB03 data where both the methods of Stawiski *et al.* Stawiski *et al.* (2003) and Szilagyi *et al.* Szilagyi and Skolnick (2006) perform significantly better. This is not surprising since we expect structure-based features to work better with limited positive data as is the case with the JMB03 dataset. In other words, our sequence-based features encode weaker characteristics requiring more data to perform as well as the stronger structure-based features. Similarly, the lower sensitivity of the ABME06 dataset can also be accounted for by the limited size of the positive (DNA-binding) set.

While nearly every subsequent dataset created in previous work has become successively larger, they do not necessarily become harder for sequence analysis techniques. With this view, we constructed two new datasets with the aim of creating the largest and

Table 6.4: Comparison with Previous Work and Blast

| | | Accuracy | MCC[a] | Sensitivity | Specificity |
|---|---|---|---|---|---|
| JMB03 | Blast | 79.3 | 21.5 | 27.8 | 90.4 |
| | AdaBoost | 89.1 | 66.2 | 48.1 | 98.0 |
| | Stawiski *et al.* Stawiski *et al.* (2003) | 92.0 | 74.0 | 81.0 | 94.4 |
| | Szilagyi *et al.* Szilagyi and Skolnick (2006) | - | 73.0 | - | - |
| JMB04 | Blast | 81.4 | 70.4 | 80.8 | 81.8 |
| | AdaBoost | 89.9 | 84.9 | 84.6 | 93.6 |
| | Ahmad *et al.* Ahmad and Sarai (2004) | 83.9 | 68.0 | 80.8 | 87.0 |
| | Szilagyi *et al.* Szilagyi and Skolnick (2006) | - | 79.0 | - | - |
| NAR05 | Blast | 82.4 | 70.2 | 75.2 | 86.1 |
| | AdaBoost | 94.7 | 88.8 | 88.4 | 97.9 |
| | Bhardwaj *et al.* Bhardwaj *et al.* (2005a) | 86.3 | - | 80.6 | 87.8 |
| JMB06 | Blast | 71.8 | 45.1 | 79.7 | 61.8 |
| | AdaBoost | 85.9 | 74.8 | 89.9 | 80.9 |
| | Szilagyi *et al.* Szilagyi and Skolnick (2006) | - | 74.0 | - | - |
| ABME06 | Blast | 72.7 | 32.5 | 42.7 | 83.2 |
| | AdaBoost | 89.6 | 74.3 | 69.3 | 96.7 |
| | Langlois *et al.* Langlois *et al.* (2007) | 88.5 | - | 66.7 | 96.3 |
| NEW07b | Blast | 72.9 | 46.3 | 59.4 | 80.4 |
| | AdaBoost | 84.0 | 69.5 | 68.8 | 92.4 |
| NEW07c | Blast | 69.4 | 28.6 | 42.6 | 82.4 |
| | AdaBoost | 84.7 | 66.2 | 64.8 | 94.4 |

[a] Maximum Matthew's Correlation Coefficient from ROC curve.

most difficult to date. This is important because machine learning algorithms perform better with more unbiased data; *i.e.* they learn more general rules. Furthermore, a larger amount of data allows better estimates of a learning algorithms ability to generalize the data. Firstly, the results over the NEW07b dataset demonstrate that our protocol loses little generalization with a larger more diverse negative set. Specifically, only about one percent is lost in terms of accuracy and area under the ROC. Secondly, since this is a sequence-based technique, we created a more rigorous positive (DNA-binding) set at 25% identity. On this dataset, accuracy and area under the ROC again only change slightly *i.e.* 84.0% versus 84.7% and 92.3% versus 91.5% AUR for the NEW07b and NEW07c, respectively. Likewise, the relative increase in the ratio between positive and negative sets of JMB06, NEW07b and NEW07c datasets results in a decreasing sensitivity and an increasing specificity. Finally, the maximum Matthew's correlation coefficient (MCC) is the highest when the imbalance in the datasets is minimized (JMB06).

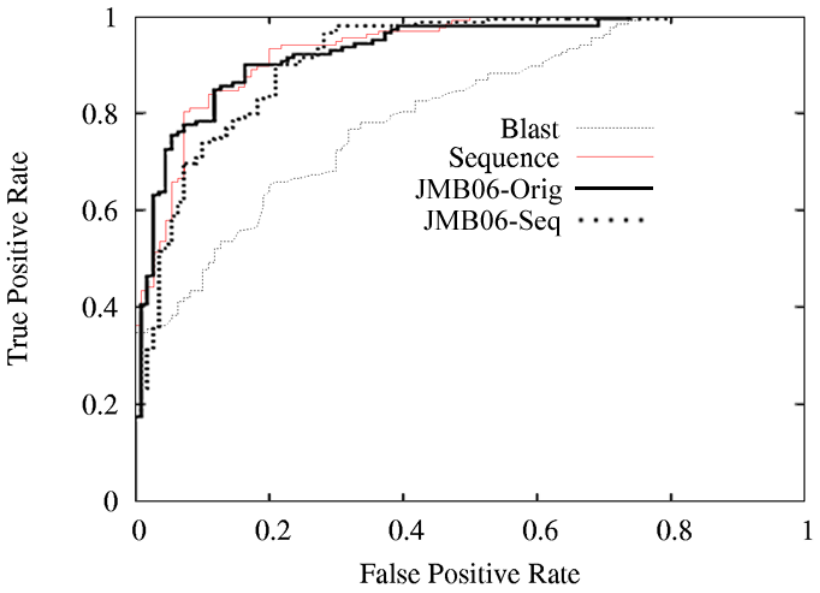


Figure 6.4: A ROC comparison of the new sequence-based feature representation and boosted trees with the JMB06 structure-based protocol and Blast.

### Comparison to Blast

Along with an extensive comparison to previous work, we also compare this method directly to a standard sequence analysis method, Blast Altschul *et al.* (1997). This comparison was performed on the same training and testing partitions for each fold of cross-validation. For this comparison, the ROC curves and area under the ROC best demonstrate that significant learning has occurred on each dataset (at a fraction of the computational cost). Specifically, the average improvement by our method over blast is about 20% in terms of area under the ROC on all but two datasets (JMB04, NAR05). The corresponding ROC curves for the JMB06 dataset in 6.4 and the ABME06 dataset in 6.3 demonstrate a similar improvement in our method over Blast.

### Knowledge Learning

While the boosted tree classifier performs best, the analysis of its model (as well as that of SVM) remains an open problem. In contrast, the ADTree classifier both performs well and has an interpretable model. By analyzing the ADTree model, we can gain insights into the relationships between the characteristics that define protein-DNA interactions. Firstly, we will define a set a criterion to judge the relevance of a propensity rule in the context of the model. Secondly, we will examine individual features and their relationships using this criterion. Thirdly, we will use two groups of DNA-binding proteins, helix-turn-helices

and nucleases, to understand the biological relevance of this knowledge.

### Feature Importance

There are four criterion used to identify reasonable rules. Firstly, the robustness of a rule is determined by bootstrapping the model 1000 times and counting the number of times a rule appears in each bootstrapped model. This count is represented on the tree by a color code (see 6.5). Secondly, the quality of a rule depends on the class of interest and the problem domain. For DNA-binding proteins, we are interested in rules that reflect the properties of the protein that facilitate binding DNA. The third and fourth criterion, confidence and support, help define the context of a rule in the model. That is, the confidence measures the amount influence the rule has in the model and should favor the class of interest. Unlike confidence, support measures how well this rule generalizes the data independently of all other rules.

The ADTree model in 6.5 comprises, atypically, three distinct trees that additively contribute to the entire model. This ADTree model achieves 87% AUR with both 10-fold cross-validation and bootstrapping 1000 models over the JMB06 dataset. Two of the tree trees are rooted with the conserved features charge and a number arginine found in an environment that prefers a beta sheet structure. The third root is feature that checks for a large number of alanine found in an environment that prefers a sheet structure; this feature predicts proteins bind DNA only if they do not contain this characteristic. Indeed, less than half of the DNA-binding proteins fall in this category while more than half of the non-binding proteins do not. Thus, this rule is not conserved, has low support and is not interesting; this learned rule may be an artifact of a small dataset.

The internal nodes of the ADTree mainly consist of relevant features with the probable exceptions of both the conditional glycine and aspartate found in the rightmost tree. Like the conditional alanine feature mentioned earlier, both of these features predict the negative class with their abundance and thus they could be artifacts of the small dataset. At the same time, the relevant features include: charge, arginine, lysine, leucine and histidine, all known to be important in DNA-binding. This suggests the model largely captures relevant information about the data. Moreover, this result is similar to yet expands upon the result in Szilagyi *et al.* Szilagyi and Skolnick (2006).

### DNA-binding Subgroups

One defining characteristic of DNA-binding proteins according to both prior knowledge and the model in 6.5 is the total charge on the protein. That is, 88 of 138 DNA-binding proteins have a significant charge (>2.6) while only 13 of the 110 non-binding proteins have this same characteristic. Of the DNA-binding subgroups, this characteristic is found in a majority of each subgroup except hydrolases and transferases. The negatively charged DNA-binding proteins generally fall into two categories: those requiring some ligand and

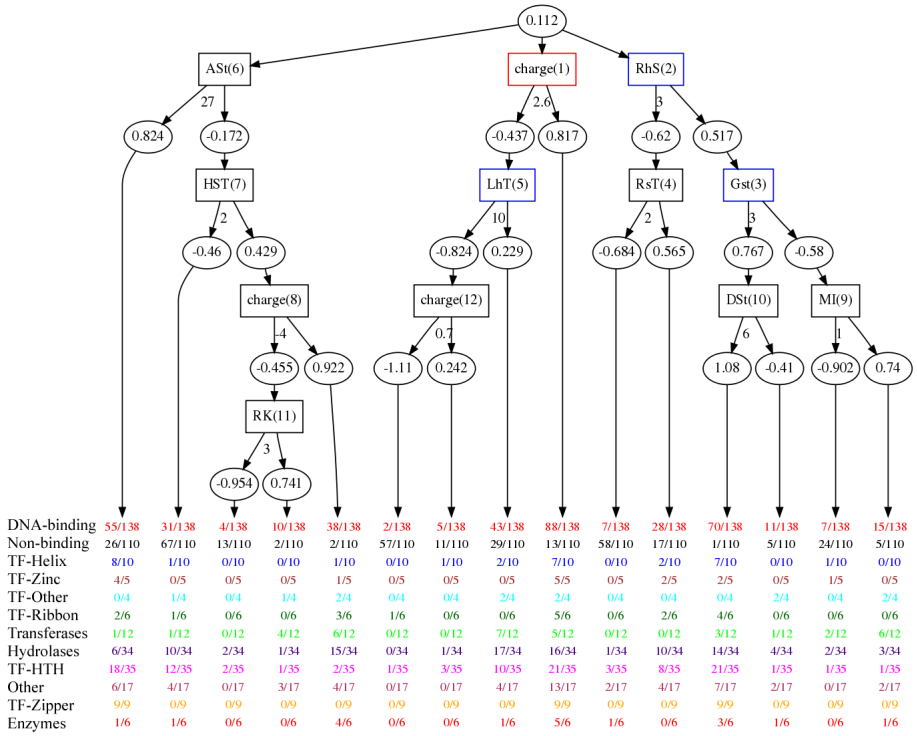


Figure 6.5: An ADTree built over the JMB06 dataset. The square nodes in the model hold the name of the feature and order it was learned. The round nodes hold the weighted vote where a positive number predicts DNA-binding. Below the square node is the threshold of prediction, if this number is exceeded then the right path is taken, otherwise the left. Below each path in the tree, there is a set of numbers in the format counted/total for the prefixing DNA-binding subgroup.

those that have a slightly positive charge. Most nucleases fall toward the negative side of the charge spectrum; this is consistent with known mechanisms of function and regulation. For example, many restriction enzymes require metal ions ($Mg^{2+}$ or $Ca^{2+}$) to interact with DNA and are regulated by ion concentration. In the case of *Mun*I (1D02) it has been observed that a calcium ion is required to reduce electrostatic repulsion Pingoud and Jeltsch (1997). Nevertheless, every example of zinc-binding and zipper type motifs is positively charged indicating that structure and charge are sufficient in these cases alone to characterize such proteins as DNA-binding. Another important subrule is **LhT(5)** which captures leucine residues that make non-specific contacts for negatively charged nucleases *e.g.* 1DDN Luscombe and Thornton (2002). Since this feature is learned over negatively charged proteins, it represents such nucleases rather than (more intuitively) the leucine zippers.

A third important subrule follows the path from **ASt(6)** to **HST(7)**. The alanine rule subdivides proteins with large flexible regions from other less flexible proteins. DNA-binding enzymes tend to have larger flexible regions as well as histidine. This particular type of histidine serves two purposes in nucleases. Firstly, it serves as a catalytic residue (*e.g.* mutating His85 in *Pvu*II destroys cleavage activity Nastri *et al.* (1997)). Secondly, the histidine serves to coordinate metal ions (*e.g.* in *Eco*RV (1EON) Horton *et al.* (2000) and even in the helix-turn-helix the diphtheria toxin repressor (1DDN) White *et al.* (1998)). Nevertheless, a majority of the helix-turn-helix proteins and a number of nucleases do not have this type of histidine. For example, repair enzymes do not require these histidines for catalysis or ion coordination instead relying on indirect readout or recruitment Vassylyev *et al.* (1995). Moreover, the restriction enzymes and DNAase I (2DNJ) still use histidine to coordinate metal ions Lahm and Suck (1991); however, this histidine is found in other types of environments.

The final subrule starts from **RhS(2)** to **RsT(4)**. These rules test for a specific types of arginine possessed by nearly every type of DNA-binding protein. For instance, this arginine distinguishes the repair enzyme, T4 endonuclease V (1VAS), from the other positively charged nucleases. This is consistent with the view that arginine is more important for site specific recognition in direct readout whereas this repair enzyme uses indirect readout to find a lesion on the DNA Vassylyev *et al.* (1995).

### 6.3.2 Discussion

We have developed a fast method that uses boosted trees to accurately identify DNA-binding proteins; all the features are efficient to calculate. This method does not require a three-dimensional structure but it does require some knowledge of the domain boundary. Our analysis of the features found that arginine, histidine and leucine in a specific environment propensity are important in recognizing certain DNA-binding proteins. Unlike previously publish works, we believe the glycine/alanine content of the protein either reflects some artifact arising from the limited amount of data or a useful rule to subdivide classes of DNA-binding proteins for further, more specific classification. We also take a critical look at negative DNA-binding proteins whose charge prevents premature activity until some ligand is bound.

Our analysis of the features in the context of the model related to our own DNA-binding subgroups shows that leaving out certain groups for testing would would not affect the performance of this method. Unlike Szilagyi and Skolnick (2006), we choose subdivide the DNA-binding proteins by function then by structure (in the case of transcription factors). Even in this more rigorous subdivision, we find that most of the rules learned work across subgroups rather than specifically for certain subgroups. This is due to the fact that such features capture underlying mechanisms of function rather than function itself. That is, certain transcription factors as well as enzymes require ions to bind DNA hence their overall negative charge. Due to a few mistakes in the categorization of DNA-

binding proteins by Szilagyi *et al.* Szilagyi and Skolnick (2006), its hard to judge how well their method would work on novel folds. However, this work supports their conjecture that these methods can in fact recognize novel folds.

We also directly compare our method to a standard sequence analysis technique, Blast, and over five datasets from previously published works. We show that our method performs as well as previous works and significantly better than blast on proteins sets with little sequence similarity. Specifically, our sequence-based method performs competitively with published methods that rely (in a few cases) on expensive sequence and structure feature calculations. We found that a possible drawback to our method would be a lack of positive examples, *e.g.* JMB03 dataset. However, with the continually growing repositories, this should not pose a significant problem. Thus, our method has significant advantages over previous work. First, our method is robust to inaccurate structures or to the presence/absence of DNA in the structure. Second, since our method is both efficient and accurate it can be applied to large scale structural genomics studies. Finally, since our method is both intuitive, simply structured yet non-linear, we can thoroughly investigate and fully make use of current and prospective features in the context of the model.

## 6.4 Residue-DNA Identification

The functional site on a protein is both necessary and sufficient to determine the function of that protein (or at least one of its functions). Locating a functional site facilitates mutagenesis studies, one painstaking experimental method to determine a binding site. Indeed, given the time consuming nature of these experiments, an automated protocol for annotation would be particularly useful. The following work of Bhardwaj *et al.* Bhardwaj *et al.* (2005b) utilizes a support vector machines (SVM) classifier to build a model from a set of training examples.

### 6.4.1 Methods

The DNA-binding residue prediction problem can be approximated by classification where residues comprise examples and binding status as the label. The dataset comprises residues from 115 protein-DNA crystallographic complexes better than 3 Å; this is a union of previous datasets Ahmad *et al.* (2004); Stawiski *et al.* (2003) where the proteins span various structural families. Unlike these previous studies Ahmad *et al.* (2004); Stawiski *et al.* (2003), only surface residues are considered for the negative set since buried residues can be immediately assigned non-binding. A residue is considered on the surface if more than 40% of its area is exposed and a surface residue is considered binding if any atom fell within a distance of 4.5Åfrom DNA.

A residue and its environment determine its DNA-binding potential. Indeed, a classifier requires an example to be represented by a set of numerical features; thus, the

Table 6.5: Compares SVM on DNA-Residue Prediction

| | Accuracy | Sensitivity | Specificity | Net Accuracy |
|---|---|---|---|---|
| LOOCV-Residue[a] | 79.0 | 59.0 | 85.0 | 70.0 |
| LOOCV-Protein | 66.0 | 43.0 | 81.0 | 63.0 |
| Holdout-Protein | 75.0 | 40.0 | 81.0 | 65.0 |

[a] LOOCV: leave-one-out cross-validation

residue-environment pair is represented by the following features:

1. Charge on each residue assigned by CHARMM20 Brooks *et al.* (1983)
2. Average potential on each residue as calculated by Delphi Gilson *et al.* (1988)
3. Secondary structure of residue as calculated by DSSP Kabsch and Sander (1983)
4. Solvent accessible surface area as calculated by DSSP
5. Residue neighbors within 4.5 Å

### 6.4.2 Results

To assess the ability of SVM to generalize the residue training data, several validation techniques were used including leave-on-out cross-validation (LOOCV) on the residue level, LOOCV on the protein level and hold out on the protein level. Each of these techniques tests a specific aspect of the SVM classification protocol. LOOCV on the residue level tests whether more data will significantly increase a prediction. LOOCV on the protein-level illustrates how the performance of this protocol holds up in a realistic situation where an unseen protein provides the test residues. Finally, the hold out tests the performance of the protocol on limited data.

The results shown in 6.5 compare the performance of our protocol over the three aforementioned validation methods. The accuracy of our method is 79% similar to previous work Ahmad *et al.* (2004). However, compared to previous datasets Ahmad *et al.* (2004); Stawiski *et al.* (2003), the current class skew was reduced from 1:10 to 1:5. Specifically, internal residues are easily identified as non-binding unfairly biasing previous results. In addition, the sensitivity of our method improves by 19% sensitivity to 59% over previous work Ahmad *et al.* (2004). This corresponds to a 11% increase in net accuracy.

On the protein level, each of the metrics drops. This drop most significantly occurs in terms of sensitivity. Indeed, the specificity, remains somewhat constant even on the protein level. This implies that the decrease in the number of examples most significantly

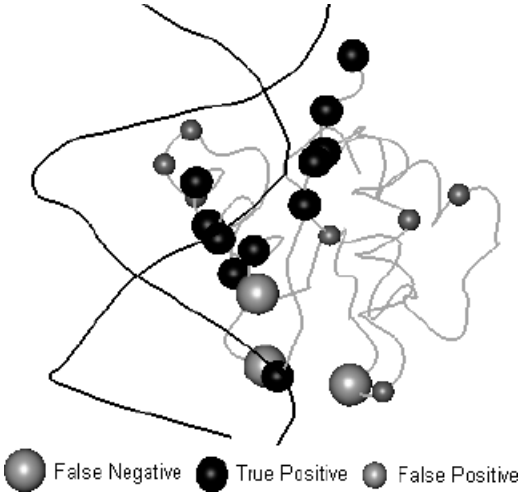


Figure 6.6: Prediction of the binding residues on 1PUE as an example. DNA and protein are shown in the black and gray tube representation.

affects sensitivity, the accuracy of identifying DNA-binding residues. This makes intuitive sense given the class skew; *i.e.* there are five times fewer DNA-binding residues to non-binding residues.

For illustrative purposes, this protocol is applied to one left out protein, a Pu.1 ETS domain (1PUE) in 6.6. It represents a typical case where residues are predicted with 70% accuracy. One observation from 6.6 is that many false positives fall in isolated locations along the structure. Thus, some post-processing could potentially improve the identification of DNA-binding residues.

### 6.4.3 Discussion

This work on DNA-binding residue prediction investigates a more realistic definition of the problem. Specifically, only surface residues are considered, making the performance less biased. Moreover, the validation is performed on the protein level. However, this formulation of the DNA-binding residue prediction as a classification task fails to leverage all the available information. In future work, we will discuss better formulations of the residue binding identification problem.

## 6.5 Future Work

In future work, we plan to incorporate one more protocol into our general DNA-binding prediction scheme. This prediction protocol will identify protein binding sites on DNA.

By combining these predictors together, we can create a hypothetical transcription regulation network. In other words, we will identify all DNA-binding residues, model their structures, determine their binding sites and finally dock the protein to the DNA to determine its binding sites on the DNA. Before this final work can be achieved we must first improve the performance of the DNA-binding protein and residue prediction. Given the computational demands of structural modelling, a protein or its domain must be first predicted to bind DNA. Several important steps have been taken to perform DNA-binding protein prediction. Most of the future work will comprise developing a better feature representation and combining a number of classifier for maximum predictive accuracy. Still several problems remain for DNA-binding residue prediction.

While classification is a convenient problem formulation for DNA-binding residue prediction, there are a number of issues that classification cannot address. The first problem is the definition of DNA-binding for the residue. One simple solution is to weight each example based on the average distance or average number of atoms in contact with DNA. This weight can be leveraged by an importance-weighted classifier, which attempts to make less mistakes on more important residues. In addition, residues that bind DNA do not act alone and thusly are not independent in terms of DNA-binding residue prediction. Thus, a structured-learning problem would be a better formulation rather than simple classification. That is, a structured-learner will search for and utilize the dependencies (or structure) between examples along

# Chapter 7

# Comparative Proteomics: DNA-binding

In the previous chapter I introduced a machine learning method to identify DNA-binding proteins from sequence alone. This method was tested over known DNA-binding domains having PDB structures. In this chapter, I first generalize this protocol and demonstrate that it works well on full protein sequences even though each sequence may have many domains that have nothing to do with DNA-binding Langlois and Lu (2007c). Then, I apply this protocol to genome-wide prediction in order to analyze how features invariant (to some degree) to sequence similarity characterize DNA-binding across genomes.

## 7.1 Background

Since the completion of the human genome project, there has been a concerted effort to find all the functional genome sequences and use this knowledge to improve the health of an individual. One such effort concerns the comparative analysis of genomes Hardison (2003). One such comparative analysis utilizes a phylogenetic tree to assign functions genome wide Pellegrini *et al.* (1999) (see 7.1). This analysis has yielded considerable insight into basic genomic features. For example, observations drawn from comparing mammalian genomes suggest that large sections of the mammalian chromosome have a tendency to change through one of several processes including single nucleotide substitution, transposon insertion and recombination Consortium (2002). Likewise, the comparison between Human, Old World and New World monkeys has also been used to capture common mammalian and primate-specific functional elements in the human genome Boffelli *et al.* (2003).

Comparing genomic sequences only gives board insights into the underlying process of evolution on the sequence level. However, the overall selection of these changes and their effect can only be seen on a higher level. These studies have been generalized to a

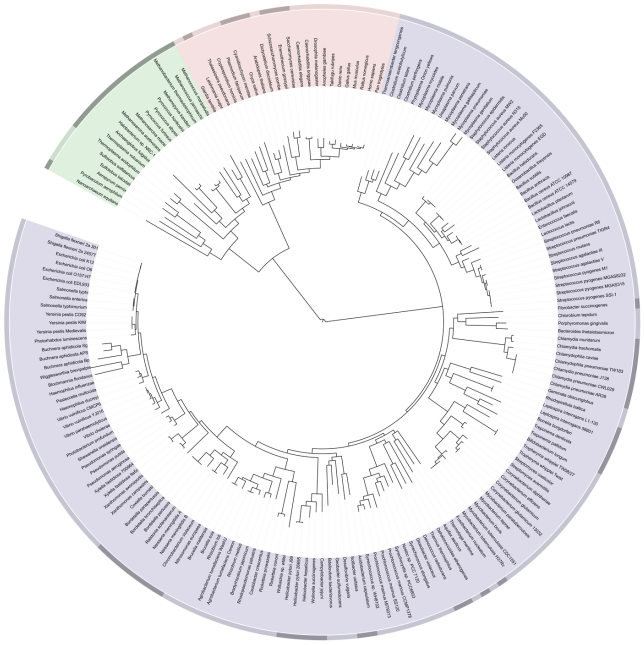

Figure 7.1: This tree compares 191 species over the three domains of life Ciccarelli *et al.* (2006). The image was created by created by Interactive Tree of Life (iTOL)Letunic and Bork (2007).

comparison of gene-coexpression in microarrays across genomes with significant amounts of data including Homo sapiens, Drosophila melanogaster, Caenorhabditis elegans, and Saccharomyces cerevisiae Stuart *et al.* (2003). Indeed, this work has discovered several genetic modules such as ancient dedicated modules (ribosomal function), evolving modules (neuronal functions) and models with interchangeable parts (metagenes for transcription factors). Likewise, a genome-wide comparison of eukaryotic transcription factors found that the entire complement of transcription factors from Arabidopsis, Drosophila, C. elegans, and S. cerevisiae to be very diverse Riechmann *et al.* (2000). One mechanism driving this diversity is achieved through domain shuffling as seen by the rearrangement of domains in both sequence and structure.

Other studies have compared the evolution of transcription regulation networks. One of the first prominent works Shen-Orr *et al.* (2002) analyzed network motifs in E. coli and found several outstanding motifs compared to random, which included feed forward, single input and dense overlap motifs. This work was extended to compare transcription networks of various genomes in order to determine the evolution of network differences; one such difference arose from duplication of motifs Babu *et al.* (2004). Another natural question that has arisen from network comparison considers the transfer of certain subnet-

works between organisms using sequence similarity. Yu *et al.* Yu *et al.* (2004) determined a sequence similarity threshold sufficient to transfer protein-protein and protein-DNA interactions between organisms. More recently, specific components of the transcription regulatory network (features of transcription) haven been studied on the genome-scale comparing various yeast species Borneman *et al.* (2007).

Similar to current work, there have been a number of approaches to assign function from sequence on the genome scale using machine learning and physio-chemical properties. Each of these works relies on support vector machines and one of two physio-chemical descriptions: one that combines descriptors from several groups Ding and Dubchak (2001) and one developed by Fujishima *et al.* Fujishima *et al.* (2007). The first set of works use Ding and Dubchak's descriptor, which represents the distribution, transition and composition of the sequence in terms of both residue type and property Ding and Dubchak (2001). These methods include prediction of enzyme families Cai *et al.* (2004), lipid binding Lin *et al.* (2006), RNA-binding Han *et al.* (2004), and family assignment Cai *et al.* (2003). The second type of feature description used amino acid properties to group amino acids and measured the periodicity of these groups Fujishima *et al.* (2007). The new feature representation was used to discriminate both DNA-binding and RNA-binding proteins Fujishima *et al.* (2007).

In the work, I will investigate the use DNA-binding protein characteristics to compare select organisms on the genome-scale. First, I define an expanded version of a general set of features (discussed in Chapter 6) to characterize protein sequences Langlois and Lu (2007a,c). Second, I use a classification algorithm on a dataset of both DNA-binding and non-DNA-binding proteins to find features important to the function of binding DNA. The strength of the model holding these features is measured by its generalization performance. To this end, I have developed the a successful protocol to identify protein function on the genome scale using sequence features (not similarity) alone. Third, I compare boosted trees against blast using both within genome predictions and cross genome predictions. While sequence similarity is not invariant to evolution, our sequence-based features will be to some degree. Thus, characterizing the features that have evolved may give some insight into the features that are important for regulation and maintenance of the human genome.

## 7.2 Methods

Here, I introduce the new datasets and features developed to investigate DNA-binding proteins on the genome-scale. The following results use both training and testing sets to compare genomes and 5-fold cross-validation (introduced in Chapter 2) for within genome results. I compare the local sequence alignment algorithm, Blast (introduced in Chapter 5) with boosted decision trees (introduced in Chapter 2).

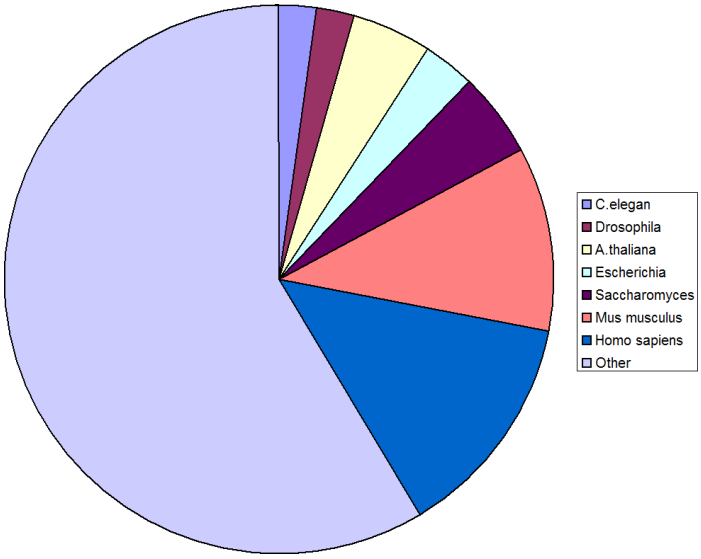


Figure 7.2: This pie graph shows the relative sequence abundance of each genome compared to the total for the original redundant subset extracted from the SwissProt.

### 7.2.1 Datasets

This dataset comprises a subset of Swiss-Prot Consortium (2007) Release 54.5 (13-Nov-2007), which contains 289,473 annotated protein sequences. The DNA-binding proteins are extracted using two fields: keywords (KW) and database cross-reference (DR). A protein is considered to bind DNA if it has one of the following in the keyword list: DNA-binding, Topoisomerase, Intron homing, DNA invertase, DNA-directed DNA polymerase or DNA-directed RNA polymerase. It is also considered DNA-binding if one of the following is found the in database cross reference: DNA-binding, TRANSFAC or transcription factor activity. A protein is considered non-DNA-binding if it is not classified as a DNA-binding protein and does not have Nucleus in the keyword field; these proteins are also restricted to one of the following compartments: Cytoplasm, Membrane, Transmembrane, Cytoskeleton, Cytoplasmic, Endoplasmic reticulum, Golgi apparatus or Periplasm. This left 20,270 sequences labeled as DNA-binding and 112,393 sequences as non-binding.

The dataset was then filtered to remove redundant sequences. This was performed separately for the DNA-binding and the non-binding subsets using the CD-HIT program Li and Godzik (2006). Indeed, since sequence comparison is a slow process, this program uses a clustering algorithm to efficiently handle large sequence datasets *e.g.* the non-binding subset has 112,393 sequences. Two subsets were created such that the first ensured that no two sequences had more than 40% sequence identity and the second 25% sequence identity. This left 28,236 and 15,611 protein sequences, respectively.

I also create single genome subsets for seven organisms: C. elegan (Worm), Drosophila

(Fly), A. thaliana (Plant), Escherichia coli or E. coli (Bacteria), Saccharomyces (Yeast), Mus musculus (Mouse) and Homo sapiens (Human). In order to reduce the dataset to 25% identity, a version of CD-HIT relying on PSI-Blast was used. This version of CD-HIT while more efficient then using PSI-Blast alone, still requires significant computational resources. Thus, for each genome, I extract sequences from the 40% (to remain consistent) and 25% subsets rather than reducing the redundancy in a pairwise fashion. This procedure has the added benefit that every genome subset has no proteins that share more than 40% or 25% between them. However, this creates substantially smaller subsets *e.g.* for E. coli creating a subset by extraction gives half as many DNA-binding proteins as reducing the redundancy from only the known E. coli proteome. 7.2 shows the relative abundance of each genome in the redundant dataset. The complete statistics of each dataset can be found in A.1 and A.2 of Appendix A.

### 7.2.2 Features

In Chapter 6, I introduced a new sequence-based feature representation called local environment amino acid composition. This feature representation encodes a protein sequence by counting the number of amino acids of a specific type that lie in a specific environment. I account for 24 amino acid types, which combines the 20 standard types and four types used when the exact type is unknown: ASX (Asparagine/Aspartate), XLE (Leucine/Isoleucine), GLX (Glutamine/Glutamate) and XXX (Unknown). Previously, I considered only secondary structure propensity as an environment; here, I consider 25 different physio-chemical properties found in the AAIndex Kawashima *et al.* (1999) including: hydrophobicity Juretic *et al.* (1998), solvation Eisenberg and McLachlan (1986), charge Brooks *et al.* (1983), helix propensity Prabhakaran (1990), sheet propensity Prabhakaran (1990), buriability Zhou and Zhou (2004), stability Zhou and Zhou (2004), interactivity Bastolla *et al.* (2005), charge2 MacKerell *et al.* (1998), hydrophobicity scales Ponnuswamy (1993), average surrounding hydrophobicity Manavalan and Ponnuswamy (1978), flexibility Bhaskaran and Ponnuswamy (1988), hydrophobicity of side-chain Jones (1975), hydrophilicity Kuhn *et al.* (1995), surface propensity Radzicka and Wolfenden (1988), graph Fauchere *et al.* (1988), polarity Radzicka and Wolfenden (1988), turn propensity Prabhakaran (1990), TOTLS Cornette *et al.* (1987), donor Charton and Charton (1983), transfer Charton and Charton (1983), isoelectric point Zimmerman *et al.* (1968), linker Suyama and Ohara (2003) and inter-domain linker Bae *et al.* (2005) (twice). I also include five sum features such as charge Brooks *et al.* (1983), helix Prabhakaran (1990), sheet Prabhakaran (1990), and hydrophobicity Juretic *et al.* (1998). The final feature vector holds 1,205 values comprising 5 sum features and 24 amino acids found in 50 environments (*e.g.* hydrophobic or not hydrophobic).

## 7.3 Results

In this section, I cover the results of this proteomics comparison. First, I analyze the proteome datasets that result after reducing their redundancy. Second, I investigate how well the boosted tree protocol compares against blast when using a portion of one proteome to predicted all other proteins in that same proteome. Here, I also test this protocol over proteins collected from many proteomes. Third, I compare learning a model built on one proteome and tested on others. Here, I demonstrate that blast does not work well while our protocol captures actual evolutionary distance between organisms. The results are reported in terms of area under the ROC curve (AUR), which measures the ability of the classifier to make both accurate and confident predictions.

### 7.3.1 Dataset Analysis

The SwissProt database (version 54.5) comprises 11,116 species of which 308 are well represented. I created eight subsets of the SwissProt at two levels of redundancy each: 40% and 25%. The first subset combines all sequences classified as binding or not binding DNA that span a number of species. This dataset has 28,236 sequences at 40% identity and 15,611 at 25% 7.3. Seven other organism specific datasets were created for Human, Yeast, Mouse, Ear cress, Drosophila, C. elegan and E. coli. The largest organism datasets are Human and Yeast for both 40% and 25%. The other datasets tend to be half the size and their relative sizes change between 40% and 25%.

### 7.3.2 Proteome Analysis

One concern in extending the boosted tree protocol to sequence is that in most cases the structural/functional domain boundaries are unknown. In other words, a DNA-binding protein sequence may have several domains only one of which binds DNA. These other domains may contribute an unacceptable amount of noise to this new sequence-based descriptor. At the same time, these multiple domains may help blast in that it matches two non-binding domains in a DNA-binding protein and still makes the correct prediction. Thus, I compare blast and this new boosted tree protocol by training and testing on proteins from the same proteome. This should litigate any evolutionary artifacts that could degrade the results of blast. Specifically, I perform a 5-fold cross-validation of each of the seven organism proteomes and the comprehensive proteome. Note, the raw data for the following experiments can be found in B.1 of Appendix B.

The results in 7.4 suggest that the new features in the boosted tree model are more invariant to sequence similarity than blast. Indeed, this new boosted tree protocol dominates blast over each dataset and performs particularly well on Yeast, almost a 15% improvement over Blast. Further, the new protocol improves by 5% over blast on Mouse, Human, C. elegan and E. coli. Likewise, these results, in most cases, improve on the

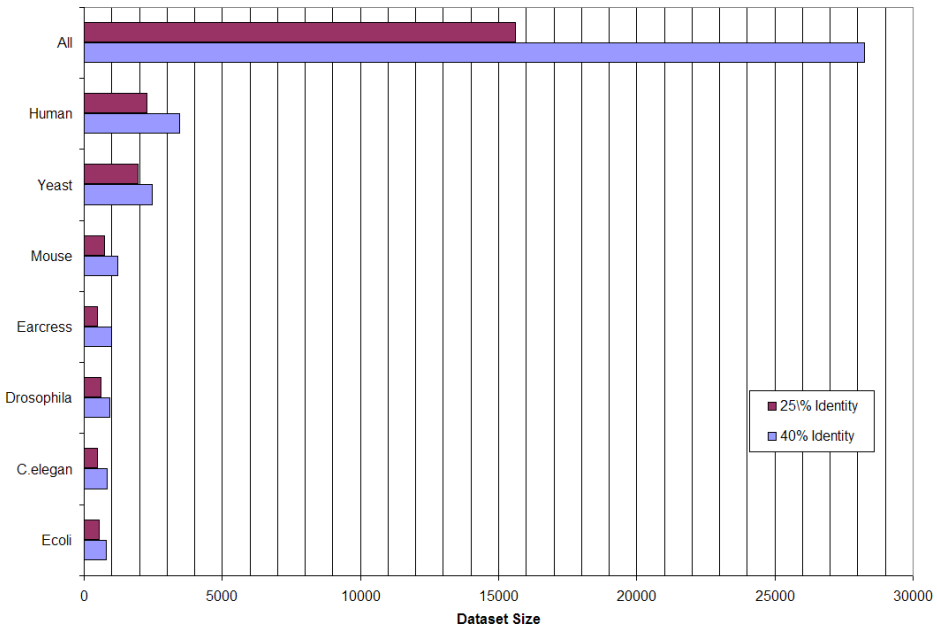


Figure 7.3: Comparison of dataset sizes for seven organisms and the SwissProt subset with redundancies of 40% and 25%.

larger 40% dataset. This negates the concern that the 25% datasets are too small for an accurate assessment. For example, in Drosophila the new protocol improves by nearly 5% from 25% to 40% where the number of examples nearly doubles. Finally, the datasets are sorted from the smallest on the left to the largest on the right. From this order, it becomes clear that both blast and boosted trees perform worse on the larger datasets. At the same time, these larger datasets represent more complex organisms (except E. coli and Yeast). Thus, it is difficult to tell if the results on the smaller datasets are overly optimistic or these organisms use a more limited/similar set of characteristics.

The results in 7.5 compare blast and boosted trees over the datasets at 40% identity. At this sequence identity, blast is able to identify more similarity between sequences and thusly performs substantially better over every dataset. This is best exemplified by the comprehensive dataset (All) were blast outperforms the boosted tree protocol by almost 5%. This result points to one weakness in using area under the ROC to measure the performance of both methods. Looking at the raw data, blast tends to make less confident predictions than boosted trees. Thus, measuring the confidence (*e.g.* using mean squared error) should change the results. In practice, these confidence values in predicting DNA-binding proteins would be considered too low for an actual assignment of function. Notwithstanding, most of the results in 7.5 favor the boosted tree protocol. That is, on six of the seven organisms the boosted tree protocol performs better and on yeast it performs substantially better with an increase of 5%. In sum, this new machine

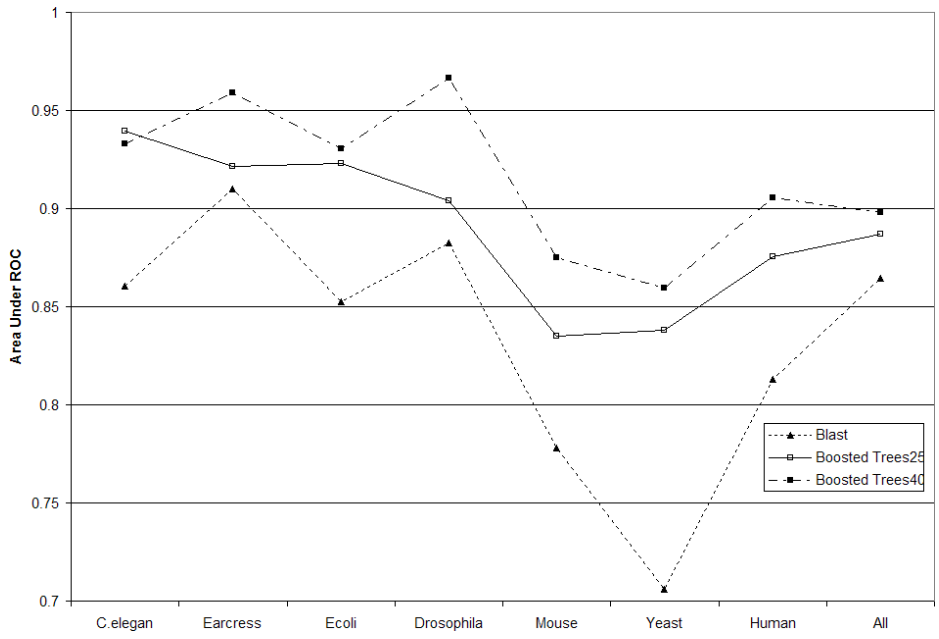


Figure 7.4: This figure compares blast and boosted trees over the seven organisms and the comprehensive dataset reduced to 25 % identity. It also compares boosted trees over both the 25% and 40% identity datasets.

learning protocol out performs blast especially when the sequence identity is low.

### 7.3.3 Proteome Comparison

By comparing the prediction of DNA-binding proteins of one organism based on a model learned from another organism one can gain insights into the relationships of both organisms. It also provides a measure of invariance of a set of features to the organism type. First, I compare the performance of a model built on a single organism to all other organisms on average. This is done first on datasets reduced to 25% sequence identity and then on 40% sequence identity. In other words, I show that this protocol is invariant to sequence similarity and to dataset size. Second, I take one of the seven organisms as a baseline and show how well this protocol performs on the other six datasets. Finally, I demonstrate that the boosted model captures evolutionary relationships of DNA-binding proteins. Note, the raw data for the following experiments can be found in C.1 through D.7 of Appendices C and D.

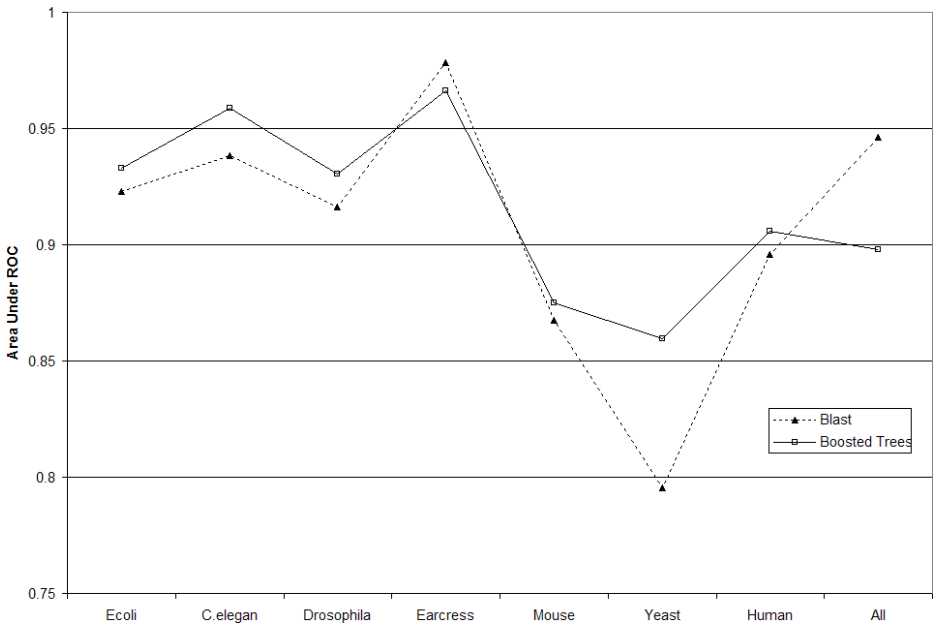


Figure 7.5: This figure compares blast and boosted trees over the seven organisms and the comprehensive dataset reduced to 40 % identity.

### Comparison of One Organism versus Others: 25% Identity

One method to compare the different DNA-binding proteins between organisms is to plot the average performance when using all other datasets to train a model then test on a single organism; Likewise, this comparison should be done by training a model on a single organism then testing on all others. Such a comparison is made in 7.6 using Boosted Trees and Blast over the seven organism datasets reduced to 25% identity. Specifically, the dark lines (-Train) compare training on the corresponding organism and testing on all others. The light lines (-Test) compare testing on the corresponding organism with models trained from all others. The last light line (Size) measures number of examples for each organism. That is, both Yeast and Human have datasets significantly larger than the rest. The standard deviation of the darker lines (-Train) is represented by the error bars. Specifically, the blast standard deviation when testing on a genome (-Test) is roughly the same in most cases. For boosted trees, the standard deviation is roughly half in nearly every case. Finally, the organisms are ordered from worst to best using the boosted tree (-Train) performance.

It is clear from 7.6 that the boost trees dominate the Blast algorithm (when comparing -Train to -Train and -Test to -Test). When training on E. coli (testing on all others), the difference between boosted trees and blast tops 17%. Likewise, when testing on E. coli, boosted trees achieve a 20% bump over Blast. This is significant because E. coli comes

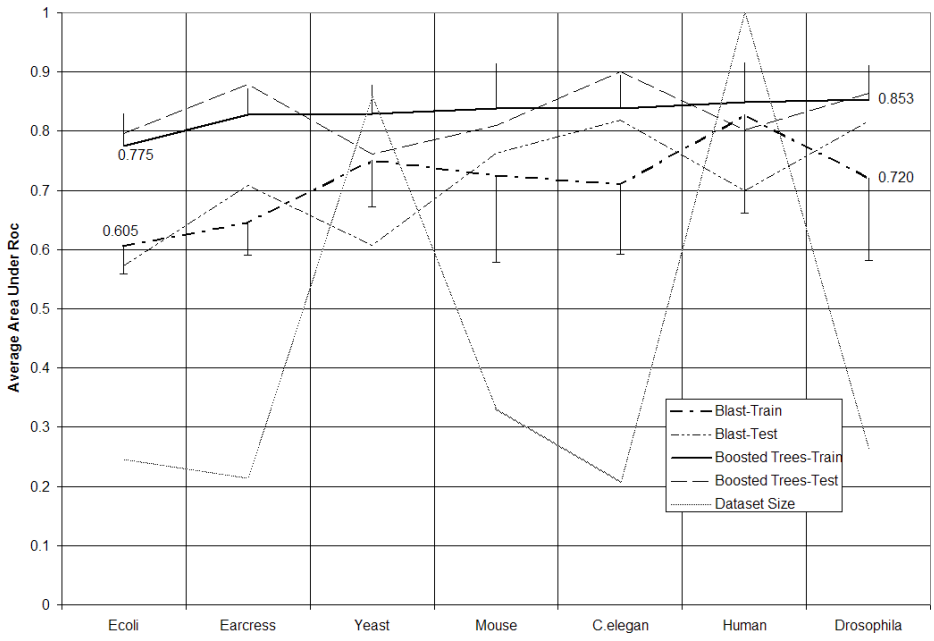


Figure 7.6: This figure compares Blast and Boosted Trees (ML) using the average area under the ROC (AUR) over the seven datasets (25% identity) (X-axis) from different organisms. The dark curves (also referred to as -Train) correspond to training on one organism and averaging the AUR over all other organisms. The lighter (-Test) curves correspond to averaging the AUR results when testing on one dataset and training on all the others. The last light line (Size) measures the size of the genome dataset for that organism. The error bars on the darker (-Train) lines show the standard deviation.

from completely different domain (kingdom) than the other organisms. Indeed, it has a good number of transcription factor domains not found in other organisms Babu *et al.* (2004). This provides further evidence that our method has found features invariant to sequence similarity.

With the possible exception of E. coli, the boosted tree classifier tends to be invariant to the organism used for training (-Train) or the number of examples 7.6. Indeed, the best result is achieved by training on one of the smallest datasets, Drosophila. This performance only fluctuates by a mere 3% across the Eukaryotic organisms. Blast, however, shows a strong correlation between database size and average performance. That is, Blast only breaks 80% area under the ROC when training on the human dataset, which is twice as large as most of the others. Likewise, it performs well on the Yeast dataset (the second largest) even though Yeast is significantly different from the other organisms.

Another manner of comparing genomes (-Test, light curves in 7.6) is to train on all other organisms and averaging the testing performance over one organism. Interestingly,

Blast and boosted trees correlate well relative to the organism where boosted trees performs 10% better in most cases. Both seem to perform better on smaller datasets since the training sets are likely larger and similarly perform worse on E. coli. Notwithstanding, while Blast performs worst on E. coli, boosted trees perform worst on Yeast. This presents further evidence that the boosted tree protocol uses more sequence invariant features that cut across domains (kingdoms).

Finally, the standard deviation of testing on all other datasets (-Train) is represented by the error bars in 7.6. As is evident from the figure, boosted trees have lower variation when testing on the various datasets. Indeed, it is expected when testing on a single dataset, this variation should be less and for boosted trees its about half. Blast, however, has a high variation in both cases (-Train and -Test) where more similar organism pairs tend to perform much better and less similar much worse. This serves to illustrate the stability of the boosted tree results again.

### One Organism versus Others Comparison: 40% Identity

In 7.7, I compare one organism versus others averaged over the 40% datasets. These datasets are generally larger adding more than 1000 examples to human and doubling many of the other datasets. However, the greater identity should give Blast a greater advantage over boosted trees. Since both graphs are ordered by boosted tree (-Train) performance, that the order remains roughly the same with only a few pairwise swaps *e.g.* Human/Drosophilia, C.elegan/Mouse, and Yeast/Earcress.

Not surprisingly, Blast does much better on the 40% dataset (7.7). In general, the Blast results are more stable paralleling boosted trees with an increase in performance of up to 6% in some cases; however, it still has a high standard deviation. It also has the same dependence on database size as in 7.6. Similarly, the boosted trees show little correlation with dataset size and still consistently outperform Blast. The increase in sequence identity increases the boosted tree performance on every dataset by about 3% in terms of AUR. Finally, the E. coli data remains the toughest for blast where boosted trees perform about 20% in terms of AUR.

### Yeast versus All Comparison

In 7.8, I compare Yeast versus others over the 25% datasets. In terms of training on Yeast, blast performs comparable to boosted tree on three of the six datasets. On the other three, boosted trees performs much better *i.e.* 9% better on C. elegan and Ear cress and 20% on E. coli. The result on E. coli is interesting in that it exemplifies that this new approach works even when there is little sequence similarity. Finally, only blast shows some correlation with dataset size *e.g.* training on human gives a 10% bump in performance. At the same time, boosted trees show little correlation with dataset size *e.g.* while the mouse dataset is far smaller than human (and they are most similar in

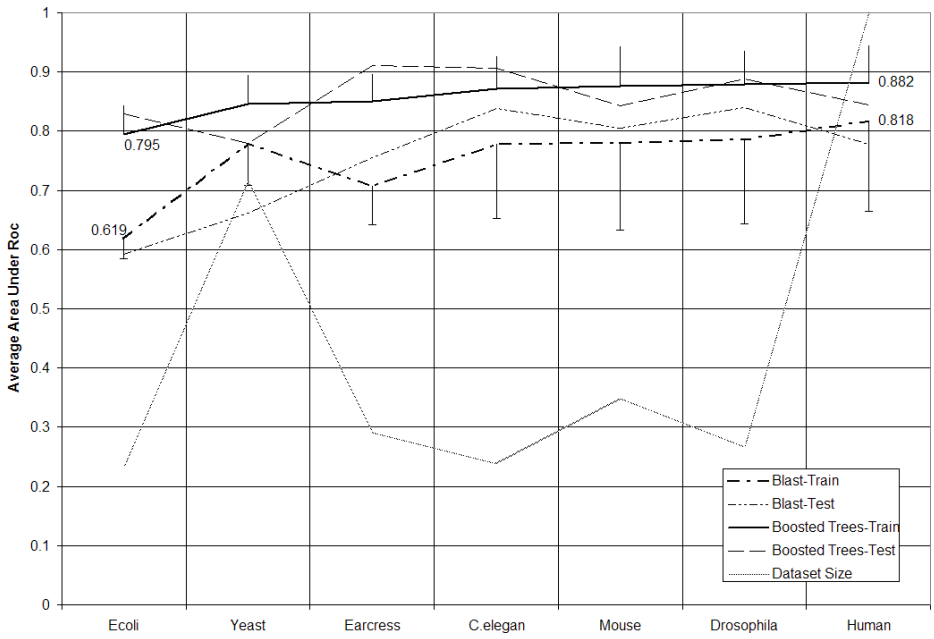


Figure 7.7: This figure compares Blast and Boosted Trees (ML) using the average area under the ROC (AUR) over the seven datasets (40% identity) (X-axis) from different organisms. The dark curves (also referred to as -Train) correspond to training on one genome and averaging the AUR over all other genomes. The lighter (-Test) curves correspond to averaging the AUR results when testing on one dataset and training on all the others. The last light line (Size) measures the size of the genome dataset for that organism. The error bars on the darker (-Train) lines show the standard deviation.

terms of sequence) there is only a 2% difference in performance when training on human and mouse. Indeed, the best performances correspond to organisms most similar to Yeast *e.g.* Ear cress, C. elegan and Drosophila (as evidenced by training on yeast and testing on them with blast).

### Phylogenetic Comparison

On natural way to compare the relative similarity of proteomes from various organisms comes in the form of a phylogenetic tree. The trees shown in 7.9 (A,B,D,E) are all cladograms grown using the neighbor joining method Saitou and Nei (1987) in SplitsTree Huson and Bryant (2006). The distance between DNA-binding proteomes is measured by the negative log of the area under the ROC averaged over the train/test pairs. Note, in a cladogram the branch lengths are unimportant concentrating instead on the relationships. In a phylogenetic cladogram, one organism forms the outgroup: a group different from the others giving perspective to the rest of the tree. Escherichia coli (E. coli) serves as

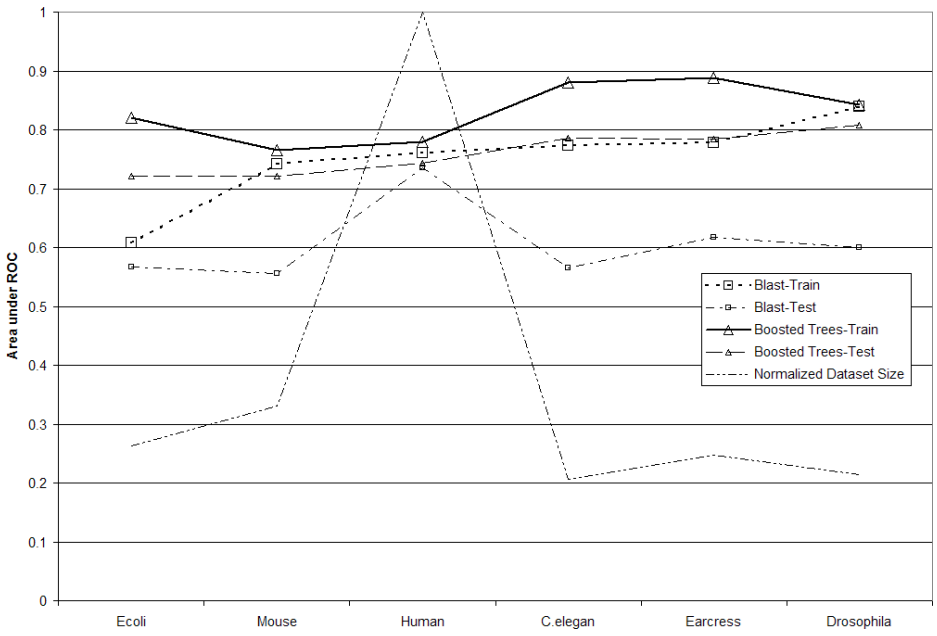


Figure 7.8: This figure compares Yeast to the six other organisms using blast and boosted trees. Each dataset is reduced to 25% sequence identity. The darker curves correspond to Yeast as the training set and the lighter ones where Yeast is the test set. The final line shows the normalized size of each dataset.

the outgroup for the cladograms in 7.9.

The top two cladograms in 7.9 compare Blast over the organism datasets at 25% (A) and 40% (B) identity. Given the low identity of the datasets, Blast is not expected to perform well. These results are compared against a standard, the center cladogram (C) built on genetic data from the interactive Tree of Life (iTOL) Letunic and Bork (2007). Specifically, both trees fail to capture the clades found in the iTOL cladogram. However, of the two, the one built on the dataset with 40% identity begins to resemble the iTOL in terms of Yeast (Saccharomyces cerevisiae) and (Mouse) Ear Cress (Arabidopsis thaliana) having a common ancestor. This result parallels another study that suggests the Ear cress proteins are more similar to animals than yeast Gutiérrez *et al.* (2004); Mayer *et al.* (1999). However, this could be a spurious result since blast fails to capture other known relationships.

The bottom two cladograms in 7.9 compare the boosted tree protocol over organism datasets at 25% (D) and 40% (E) identity. For both trees, the relationships are identical demonstrating that this protocol is more invariant to sequence similarity. Moreover, most of the clades match that of the iTOL (C) tree. The one difference originates from the swap of Arabidopsis thaliana (Plant) and Saccharomyces cerevisiae (Yeast). Interestingly, the

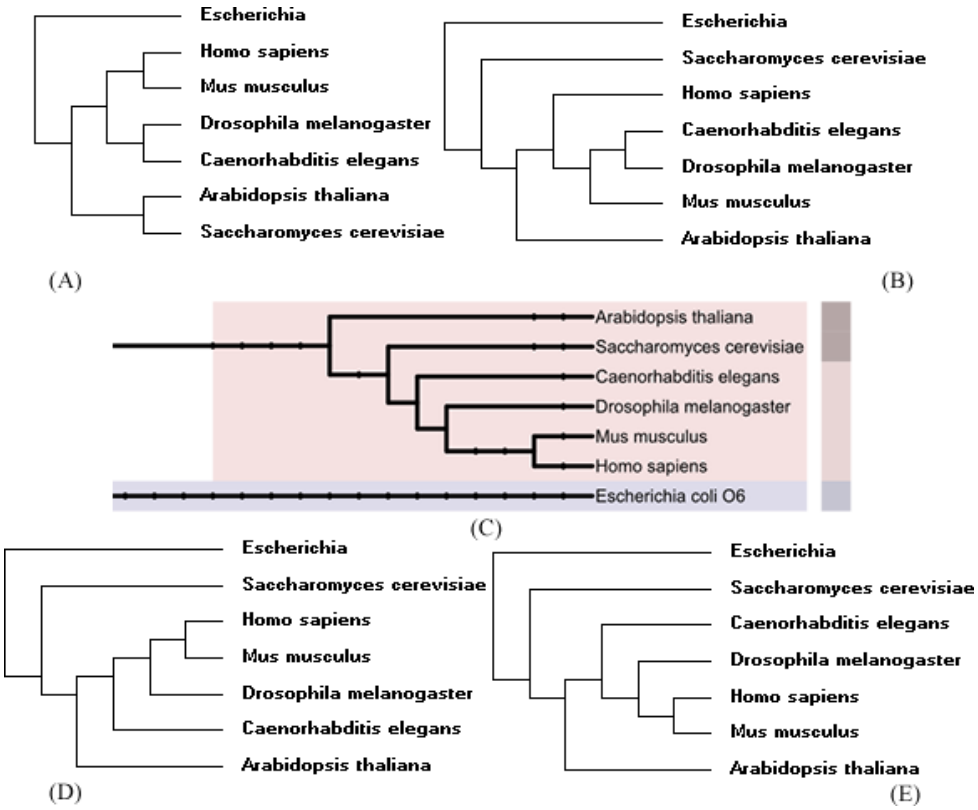


Figure 7.9: This figure compares four phylogenetic trees built using similarity of DNA-binding proteins as defined by $-\ln(AUR)$. The graphs were built on Blast on 25% (A) Blast on 40% (B) interactive tree of life (C) boosted trees on 25% (D) and boosted trees on 40% datasets.

cladograms in D and E of Figure 7.9 suggest that E. coli and Yeast, both microorganisms, have the same ancestor in terms of DNA-binding proteins. These results support evidence found in another set of studies Gutiérrez *et al.* (2004); Mayer *et al.* (1999) that also suggests that fungal (yeast) and plants split much earlier from animals. In this study, they found many proteins paralleled animal proteins more so than yeast. Moreover, this same study Gutiérrez *et al.* (2004) found that DNA-binding proteins are more unique to plants in terms of sequence similarity than any other category. This indicates that since our features are geared toward DNA-binding proteins then they must be invariant to sequence similarity.

## 7.4 Discussion

In this chapter, I expanded on a new descriptor (proposed in Chapter 6) and demonstrated that using a robust classifier (boosted trees) the resulting protocol can accurately annotate DNA-binding proteins on the genome-scale. Given the harder nature of the sequence data having multiple domains, the results still prove competitive to those on the domain level. This is clearly illustrated with a comparison to the popular sequence alignment tool, blast. I also show that this new protocol is more insensitive to evolutionary sequence differences yet captures accurate evolutionary relationships of DNA-binding characteristics.

I propose a modified evolutionary cladogram that is supported by previous experimental data. Specifically, that Yeast diverged earlier than Ear cress using DNA-binding characteristics. Moreover, on the sequence level, the DNA-binding proteins belonging to Ear cress have no relationship to animal DNA-binding proteins. However, our results indicate that the DNA-binding characteristics are more greatly shared than previous thought.

One caveat, which must be investigated in future work, is the validity of the blast results reported. While in classification terms, blast does well, the confidence values reported may not lead to a useful (confidence) prediction. Thus, it would make sense to measure these confidence values. One simple metric would be the mean squared error, which measures the average distance between a probabilistic confidence and one.

Likewise, in one future direction, one could investigate using all genomes but one in the training set as this could improve predictions on a single genome. It is interesting to consider whether more genomes will just add noise or help pick out certain classes of proteins. Likewise, it would be interesting to analyze the features that characterize DNA-binding proteins of a specific genome and more importantly, which characteristics are used when predicting another genome. This will give insights into which features are under positive and negative selection.

One final direction aims to build better datasets. First, one could design a better set of keywords to remove potential DNA-binding proteins in the negative set. That is, while DNA-binding proteins do their work in the nucleus, they still spend a great deal of time in the cytoplasm. Thus, there could be DNA-binding proteins in our negative set since being outside the nucleus is essentially the only requirement currently used. Second, one could reduce the redundancy of the datasets with more care. Using the current method of reducing redundancy is efficient but it removes twice as many examples (in some cases) as it should. A larger dataset is important for both creating a better model and getting a better estimate of the performance.

# Bibliography


Ahmad, S. and Sarai, A. (2004). Moment-based prediction of DNA-binding proteins. *Journal of Molecular Biology*, **341**(1), 65–71.

Ahmad, S. and Sarai, A. (2005). Pssm-based prediction of DNA-binding sites in proteins. *BMC Bioinformatics*, **6–12**(1), 33.

Ahmad, S., Gromiha, M. M., and Sarai, A. (2004). Analysis and prediction of DNA-binding proteins and their binding residues based on composition, sequence and structural information. *Bioinformatics*, **20**(4), 477–486.

Albright, R. A. and Matthews, B. W. (1998). Crystal structure of lambda-cro bound to a consensus operator at 3.0 angstrom resolution. *Journal of Molecular Biology*, **280**(1), 137–151.

Allwein, E. L., Schapire, R. E., and Singer, Y. (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. In *International Conference on Machine Learning*, volume 17, San Francisco, CA, USA. Morgan Kaufmann.

Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucleic Acids Research*, **25**(17), 3389–3402.

Andrews, S. and Hofmann, T. (2003). Multiple instance learning via disjunctive programming boosting. In *Advances in Neural Information Processing Systems*, volume 17, Vancouver, Whistler, Canada.

Auer, P. and Ortner, R. (2004). A boosting approach to multiple instance learning. In *European Conference on Machine Learning*, volume 15, Pisa, Italy.

Ayer, M., Brunk, H., Ewing, G., Reid, W., and Silverman, E. (1955). An empirical distribution function for sampling with incomplete. *The Annals of Mathematical Statistics*, **26**(4), 641–647.

Babu, M. M., Luscombe, N. M., Aravind, L., Gerstein, M., and Teichmann, S. A. (2004). Structure and evolution of transcriptional regulatory networks. *Current Opinion in Structural Biology*, **14**(3), 283–291.

Bae, K., Mallick, B. K., and Elsik, C. G. (2005). Prediction of protein interdomain linker regions by a hidden Markov model. *Bioinformatics*, **21**(10), 2264–2270.

Balas, E. (1985). Disjunctive programming and a hierarchy of relaxations for discrete optimization problems. *SIAM Journal on Algebraic and Discrete Methods*, **6**(3), 466–486.

Baldi, P., Brunak, S., Chauvin, Y., Andersen, C. A. F., and Nielsen, H. (2000). Assessing the accuracy of prediction algorithms for classification: An overview. *Bioinformatics*, **16**(5), 412–424.

Bao, L. and Cui, Y. (2005). Prediction of the phenotypic effects of non-synonymous single nucleotide polymorphisms using structural and evolutionary information. *Bioinformatics*, **21**(10), 2185–2190.

Bastolla, U., Porto, M., Roman, H. E., and Vendruscolo, M. (2005). Principal eigenvector of contact matrices and hydrophobicity profiles in proteins. *Bioinformatics*, **58**(1), 22–30.

Berman, H. M., Olson, W. K., Beveridge, D. L., Westbrook, J., Gelbin, A., Demeny, T., Hsieh, S. H., Srinivasan, A. R., and Schneider, B. (1992). The nucleic acid database: A comprehensive relational database of three-dimensional structures of nucleic acids. *Biophysical Journal*, **63**(3), 751–759.

Beygelzimer, A., Dani, V., Hayes, T., Langford, J., and Zadrozny, B. (2005a). Error limiting reductions between classification tasks. In *International Conference on Machine Learning*, volume 49–56, Bonn, Germany. ACM.

Beygelzimer, A., Langford, J., and Zadrozny, B. (2005b). Weighted one against all. In *Association for the Advancement of Artificial Intelligence*, Pittsburgh, Pennsylvania.

Beygelzimer, A., Kakade, S., and Langford, J. (2006). Cover trees for nearest neighbor. In *International Conference on Machine Learning*, volume 148, pages 97–104, Pittsburgh, Pennsylvania. ACM.

Beygelzimer, A., Langford, J., and Ravikumar, P. (2007). Multiclass classification with filter trees. `http://hunch.net/~jl/projects/reductions/mc_to_b/invertedTree.pdf`.

Bhardwaj, N. and Lu, H. (2007). Residue-level prediction of DNA-binding sites and its application on DNA-binding protein predictions. *FEBS Letters*, **581**(5), 1058–1066.

Bhardwaj, N., Langlois, R. E., Zhao, G., and Lu, H. (2005a). Kernel-based machine learning protocol for predicting DNA-binding proteins. *Nucleic Acids Research*, **33**(20), 6486–6493.

Bhardwaj, N., Langlois, R. E., Zhao, G., and Lu, H. (2005b). Structure based prediction of binding residues on DNA-binding proteins. In *Proceedings of the 26th Annual International Conference on the IEEE EMBS*, Shanghai, China.

Bhardwaj, N., Stahelin, R. V., Langlois, R. E., Cho, W., and Lu, H. (2006). Structural bioinformatics prediction of membrane-binding proteins. *Journal of Molecular Biology*, **359**(2), 486–495.

Bhaskaran, R. and Ponnuswamy, P. (1988). Positional flexibilities of amino acid residues in globular proteins. *International Journal of Peptide and Protein Research*, **32**, 241–255.

Blatner, N. R., Stahelin, R. V., Diraviyam, K., Hawkins, P. T., Hong, W., Murray, D., and Cho, W. (2004). The molecular basis of the differential subcellular localization of FYVE domains. *Journal of Biological Chemistry*, **279**(51), 53818–53827.

Blum, A. and Kalai, A. (1998). A note on learning from multiple-instance examples. *Machine Learning*, **30**(1), 23–29.

Blum, A., Kalai, A., and Langford, J. (1999). Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *COLT: Computational Learning Theory*, volume 12, pages 203–208, Santa Cruz, California. ACM.

Boffelli, D., McAuliffe, J., Ovcharenko, D., Lewis, K. D., Ovcharenko, I., Pachter, L., and Rubin, E. M. (2003). Phylogenetic shadowing of primate sequences to find functional regions of the human genome. *Science*, **299**(5611), 1391–1394.

Bonneau, R., Tsai, J., Ruczinski, I., Chivian, D., Rohl, C., Strauss, C. E. M., and Baker, D. (2001). Rosetta in CASP4: Progress in ab initio protein structure prediction. *Proteins: Structure, Function, and Genetics*, **45**(S5), 119–126.

Borneman, A. R., Gianoulis, T. A., Zhang, Z. D., Yu, H., Rozowsky, J., Seringhaus, M. R., Wang, L. Y., Gerstein, M., and Snyder, M. (2007). Divergence of transcription factor binding sites across related yeast species. *Science*, **317**(5839), 815–819.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140.

Breiman, L. (2001). Random forests. *Machine Learning*, **45**(1), 5–32.

Brenner, S. E., Koehl, P., and Levitt, M. (2000). The ASTRAL compendium for protein structure and sequence analysis. *Nucleic Acids Research*, **28**(1), 254–256.

Brooks, B. R., Bruccoleri, R. E., Olafson, B. D., States, D. J., Swaminathan, S., and Karplus, M. (1983). CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal Computational Chemistry*, **4**, 187–217.

Buck, M. J. and Lieb, J. D. (2004). ChIP-chip: Considerations for the design, analysis, and application of genome-wide chromatin immunoprecipitation experiments. *Genomics*, **83**(3), 349–360.

Buhlmann, P. (2003). Bagging, subagging and bragging for improving some prediction algorithms. In M. G. Akritas and D. N. Politis, editors, *Recent Advances and Trends in Nonparametric Statistics*, pages 19–34. Elsevier, North Holland.

Bunescu, R. C. and Mooney, R. J. (2007). Multiple instance learning for sparse positive bags. In Z. Ghahramani, editor, *International Conference on Machine Learning*, volume 24, pages 105–112, Corvallis, Oregon. Omnipress.

Buntine, W. (1992). Learning classification trees. *Statistics and Computing*, **2**(2), 63–73.

Buntine, W. and Caruana, R. (1991). Introduction to IND and recursive partitioning. Technical Report FIA-91-28, NASA Ames Research Center, 10. `http://opensource.arc.nasa.gov/project/ind/`.

Cai, C., Han, L. Y., Ji, Z. L., and Chen, Y. Z. (2004). Enzyme family classification by support vector machines. *Proteins: Structure, Function, and Bioinformatics*, **55**(1), 66–76.

Cai, C. Z., Han, L. Y., Ji, Z. L., Chen, X., and Chen, Y. Z. (2003). SVM-Prot: Web-based support vector machine software for functional classification of a protein from its primary sequence. *Nucleic Acids Research*, **31**(13), 3692–3697.

Cajone, F., Salina, M., and Benelli-Zazzera, A. (1989). 4-hydroxynonenal induces a DNA-binding protein similar to the heat-shock factor. *Biochemical Journal*, **262**, 977–979.

Celis, S. and Musicant, D. (2002). Weka-parallel: Machine learning in parallel. Technical report, Carleton College Computer Science. `http://weka-parallel.sourceforge.net/`.

Chang, C.-C. and Lin, C.-J. (2001). LIBSVM: A library for support vector machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Charton, M. and Charton, B. I. (1983). The dependence of the Chou-Fasman parameters on amino acid side chain structure. *Journal of Theoretical Biology*, **102**(1), 121–134.

Cheung, P.-M. and Kwok, J. T. (2006). A regularization framework for multiple-instance learning. In *International Conference on Machine learning*, volume 148, pages 193–200, Pittsburgh, Pennsylvania. ACM.

Chevaleyre, Y. and Zucker, J.-D. (2001). Solving multiple-instance and multiple-part learning problems with decision trees and rule sets. application to the mutagenesis problem. In *Conference of the Canadian Society for Computational Studies of Intelligence*, volume 14 of *Lecture Notes in Computer Science*, pages 204–214. Springer, Ottawa, Canada.

Cho, W. (2001). Membrane targeting by C1 and C2 domains. *Journal of Biological Chemistry*, **276**(35), 32407–32410.

Cho, W. and Stahelin, R. V. (2005). Membrane-protein interactions in cell signalling and membrane trafficking. *Annual Review of Biophysics and Biomolecular Structure*, **34**, 119–151.

Chou, C.-C., Lin, T.-W., Chen, C.-Y., and Wang, A. H. J. (2003). Crystal structure of the hyperthermophilic archaeal DNA-binding protein Sso10b2 at a resolution of 1.85 angstroms. *Journal of Bacteriology*, **185**(14), 4066–4073.

Chou, P. Y. and Fasman, G. D. (1978). Prediction of the secondary structure of proteins from their amino acid sequence. *Advances in Enzymology and Related Areas of Molecular Biology*, **47**, 45–148.

Ciccarelli, F. D., Doerks, T., von Mering, C., Creevey, C. J., Snel, B., and Bork, P. (2006). Toward automatic reconstruction of a highly resolved tree of life. *Science*, **311**(5765), 1283–1287.

Collobert, R., Bengio, S., and Mariethoz, J. (2002.). Torch: A modular machine learning software library. Technical Report IDIAP-RR 02-46, IDIAP Research Institute. `http://www.torch.ch/`.

Consortium, I. H. (2003). The international HapMap project. *Nature*, **426**(6968), 789–796.

Consortium, M. G. S. (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature*, **420**(6915), 520–562.

Consortium, T. U. (2007). The universal protein resource (UniProt). *Nucleic Acids Research*, **35**(suppl1), D193–197.

Cornette, J. L., Cease, K. B., Margalit, H., Spouge, J. L., Berzofsky, J. A., and DeLisi, C. (1987). Hydrophobicity scales and computational techniques for detecting amphipathic structures in proteins. *Journal of Molecular Biology*, **195**(3), 659–685.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine Learning*, **20**(3), 273–297.

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, **2**, 265–292.

Craven, M. W. (1996). Extracting comprehensible models from trained neural networks. Technical Report CS-TR-96-1326, University of Wisconsin - Madison.

Crowfoot, D. (1935). X-ray single crystal photographs of insulin. *Nature*, **135**, 591–592.

Day, W. H. E. and McMorris, F. R. (1992). Critical comparison of consensus methods for molecular sequences. *Nucleic Acids Research*, **20**(5), 1093–1099.

Demšar, J., Zupan, B., Leban, G., and Curk, T. (2004). Orange: From experimental machine learning to interactive data mining. In *Knowledge Discovery in Databases: PKDD 2004*, volume 3202 of *Lecture Notes in Computer Science*, pages 537–539. Springer, Berlin/Heidelberg. `http://www.ailab.si/orange`.

Demiriz, A., Bennett, K. P., and Shawe-Taylor, J. (2002). Linear programming boosting via column generation. *Machine Learning*, **46**(1), 225–254.

Dietterich, T. G. and Bakiri, G. (1995). Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, **2**, 263–286.

Dietterich, T. G., Lathrop, R. H., and Lozano-Pérez, T. (1997). Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, **89**(1-2), 31–71.

Ding, C. H. Q. and Dubchak, I. (2001). Multi-class protein fold recognition using support vector machines and neural networks. *Bioinformatics*, **17**(4), 349–358.

Dobson, R., Munroe, P., Caulfield, M., and Saqi, M. (2006). Predicting deleterious nsSNPs: An analysis of sequence and structural attributes. *BMC Bioinformatics*, **7**(1), 217–226.

Domingos, P. (1997). Knowledge acquisition from examples via multiple models. In *International Conference on Machine Learning*, volume 14, pages 98–106, Cavtat-Dubrovnik, Croatia. Morgan Kaufmann.

Dubchak, I., Muchnik, I., Mayor, C., Dralyuk, I., and Kim, S.-H. (1999). Recognition of a protein fold in the context of the SCOP classification. *Proteins: Structure, Function, and Genetics*, **35**(4), 401–407.

Duffy, N. and Helmbold, D. P. (1999). Potential boosters? In *Advances in Neural Information Processing Systems*, volume 12, pages 258–264, Denver, Colorado.

Eisenberg, D. and McLachlan, A. D. (1986). Solvation energy in protein folding and binding. *Nature*, **319**(6050), 199–203.

Fan, R.-E., Chen, P.-H., and Lin, C.-J. (2005). Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, **6**, 1889–1918.

Fauchere, J., Charton, M., Kier, L., Verloop, A., and Pliska, V. (1988). Amino acid side chain parameters for correlation studies in biology and pharmacology. *International Journal of Peptide and Protein Research*, **32**, 269–278.

Fernandez-Escamilla, A.-M., Rousseau, F., Schymkowitz, J., and Serrano, L. (2004). Prediction of sequence-dependent and mutational effects on the aggregation of peptides and proteins. *Nature Biotechnology*, **22**(10), 1302–1306.

Freeman, K., Gwadz, M., and Shore, D. (1995). Molecular and genetic analysis of the toxic effect of RAP1 overexpression in yeast. *Genetics*, **141**(4), 1253–1262.

Freund, Y. and Mason, L. (1999). The alternating decision tree learning algorithm. In *International Conference on Machine Learning*, volume 16, Bled, Slovenia.

Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, volume 13, Bari, Italy.

Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *Annals of Statistics*, **28**(2), 337–407.

Fujishima, K., Komasa, M., Kitamura, S., Suzuki, H., Tomita, M., and Kanai, A. (2007). Proteome-wide prediction of novel DNA/RNA-binding proteins using amino acid composition and periodicity in the hyperthermophilic Archaeon Pyrococcus furiosus. *DNA Research*, **14**(3), 91–102.

Gilson, M. K., Sharp, K. A., and Honig, B. H. (1988). Calculating the electrostatic potential of molecules in solution: Method and error assessment. *Journal Computational Chemistry*, **9**(4), 327–335.

Goutte, C. (1997). Note on free lunches and cross-validation. *Neural Computation*, **9**(6), 1211–1215.

Gärtner, T., Flach, P. A., Kowalczyk, A., and Smola, A. J. (2002). Multi-instance kernels. In *International Conference on Machine Learning*, volume 19, pages 179–186, Sydney Australia.

Gutiérrez, R., Green, P., Keegstra, K., and Ohlrogge, J. (2004). Phylogenetic profiling of the Arabidopsis thaliana proteome: What proteins distinguish plants from other organisms? *Genome Biology*, **5**(8), R53.

Han, L. Y., Cai, C. Z., Lo, S. L., Chung, M. C. M., and Chen, Y. Z. (2004). Prediction of RNA-binding proteins from primary sequence by a support vector machine approach. *RNA*, **10**(3), 355–368.

Hardison, R. C. (2003). Comparative genomics. *PLoS Biology*, **1**(2), e58.

Hobohm, U. and Sander, C. (1994). Enlarged representative set of protein structures. *Protein Science*, **3**(3), 522–524.

Horton, N. C., Connolly, B. A., and Perona, J. J. (2000). Inhibition of EcoRV endonuclease by Deoxyribo-3'-s-phosphorothiolates: A high-resolution X-ray crystallographic study. *Journal of the American Chemical Society*, **122**(14), 3314–3324.

Hou, Y., Hsu, W., Lee, M. L., and Bystroff, C. (2003). Efficient remote homology detection using local structure. *Bioinformatics*, **19**(17), 2294–2301.

Hou, Y., Hsu, W., Lee, M. L., and Bystroff, C. (2004). Remote homolog detection using local sequence-structure correlations. *Proteins: Structure, Function, and Bioinformatics*, **57**(3), 518–530.

Humphrey, W., Dalke, A., and Schulten, K. (1996). VMD: Visual Molecular Dynamics. *Journal of Molecular Graphics*, **14**(1), 33–38.

Hurley, J. H. and Meyer, T. (2001). Subcellular targeting by membrane lipids. *Current Opinion in Cell Biology*, **13**(2), 146–152.

Huson, D. H. and Bryant, D. (2006). Application of phylogenetic networks in evolutionary studies. *Molecular Biology and Evolution*, **23**(2), 254–267.

Ie, E., Weston, J., Noble, W. S., and Leslie, C. (2005). Multi-class protein fold recognition using adaptive codes. In *International Conference on Machine Learning*, volume 22, Bonn, Germany.

Jaakkola, T., Diekhans, M., and Haussler, D. (2000). A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, **7**(1-2), 95–114.

Jaenisch, R. and Bird, A. (2003). Epigenetic regulation of gene expression: How the genome integrates intrinsic and environmental signals. *Nature Genetics*, **33**, 245–254.

Jones, D. D. (1975). Amino acid properties and side-chain orientation in proteins: A cross correlation approach. *Journal of Theoretical Biology*, **50**(1), 167–183.

Jones, S., van Heyningen, P., Berman, H. M., and Thornton, J. M. (1999). Protein-DNA interactions: A structural analysis. *Journal of Molecular Biology*, **287**(5), 877–896.

Juretic, D., Lucic, B., Zucic, D., Trinajstic, N., and Cyril, P. (1998). Protein transmembrane structure: Recognition and prediction by using hydrophobicity scales through preference functions. In *Theoretical and Computational Chemistry*, volume 5, pages 405–445. Elsevier.

Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: Pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, **22**(12), 2577–2637.

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

Kawashima, S., Ogata, H., and Kanehisa, M. (1999). AAindex: Amino Acid index database. *Nucleic Acids Research*, **27**(1), 368–369.

Kearns, M. (1993). Efficient noise-tolerant learning from statistical queries. In *ACM Symposium on the Theory of Computing*, volume 25, pages 392–401, San Diego, California, United States. ACM.

Kearns, M. and Mansour, Y. (1996). On the boosting ability of top-down decision tree learning algorithms. In *ACM Symposium on the Theory of Computing*, volume 28, Philadelphia, Pennsylvania. ACM.

Keeler, J. D., Rumelhart, D. E., and Leow, W.-K. (1990). Integrated segmentation and recognition of hand-printed numerals. In *Advances in Neural Information Processing Systems*, pages 557–563, Denver, Colorado. Morgan Kaufmann Publisher.

Kihara, D., Lu, H., Kolinski, A., and Skolnick, J. (2001). TOUCHSTONE: An ab initio protein structure prediction method that uses threading-based tertiary restraints. *Proceedings of the National Academy of Sciences*, **98**(18), 10125–10130.

Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy esimation and model selection. In *International Joint Conference on Artificial Intelligence*, Montreal, Canada.

Kohavi, R., Sommerfield, D., and Dougherty, J. (1996). Data mining using MLC++, a machine learning library in C++. In *International Conference on Tools with Artificial Intelligence*, volume 8, page 234, Toulouse, France. IEEE Computer Society.

Kohout, S. C., Corbalán-García, S., Gómez-Fernández, J. C., and Falke, J. J. (2003). C2 domain of protein kinase C alpha: Elucidation of the membrane docking surface by site-directed fluorescence and spin labeling. *Biochemistry*, **42**(5), 1254–1265.

Koppensteiner, W. A., Lackner, P., Wiederstein, M., and Sippl, M. J. (2000). Characterization of novel proteins based on known protein structures. *Journal of Molecular Biology*, **296**(4), 1139–1152.

Krawczak, M., Ball, E. V., Fenton, I., Stenson, P. D., Abeysinghe, S., Thomas, N., and Cooper, D. N. (2000). Human gene mutation database – a biomedical information and research resource. *Human Mutation*, **15**(1), 45–51.

Kruglyak, L. and Nickerson, D. A. (2001). Variation is the spice of life. *Nature Genetics*, **27**(3), 234–236.

Kuang, R., Ie, E., Wang, K., Wang, K., Siddiqi, M., Freund, Y., and Leslie, C. (2005). Profile-based string kernels for remote homology detection and motif extraction. *Journal of Bioinformatics and Computational Biology*, **3**(3), 527–550.

Kuhn, L. A., Swanson, C. A., Pique, M. E., Tainer, J. A., and Getzoff, E. D. (1995). Atomic and residue hydrophilicity in the context of folded protein structures. *Bioinformatics*, **23**(4), 536–547.

Kuznetsov, I. B., Gou, Z., Li, R., and Hwang, S. (2006). Using evolutionary and structural information to predict DNA-binding sites on DNA-binding proteins. *Proteins: Structure, Function, and Bioinformatics*, **64**(1), 19–27.

Lahm, A. and Suck, D. (1991). DNase I-induced DNA conformation: 2 angstrom structure of a DNase I-octamer complex. *Journal of Molecular Biology*, **222**(3), 645–667.

Langford, J. and Beygelzimer, A. (2005). Sensitive error correcting output codes. In *Conference on Learning Theory*, Bertinoro, Italy.

Langford, J. and Zadrozny, B. (2005). Estimating class membership probabilities using classifier learners. In *International Workshop on Artificial Intelligence and Statistics*, volume 10, Barbados.

Langlois, R. and Lu, H. (2007a). Boosting the prediction and understanding of DNA-binding proteins. *In preparation.*

Langlois, R. and Lu, H. (2007b). Classification to multiple instance learning using AdaBoost. *In preparation.*

Langlois, R. and Lu, H. (2007c). Comparative proteomics with machine learning. *In preparation.*

Langlois, R. and Lu, H. (2007d). malibu: A machine learning workbench. *In preparation.*

Langlois, R., Carson, M., Bhardwaj, N., and Lu, H. (2007). Learning to translate sequence and structure to function: Identifying DNA binding and membrane binding proteins. *Annals of Biomedical Engineering*, **35**(6), 1043–1052.

Langlois, R. E., Diec, A., Dai, Y., and Lu, H. (2004). Kernel based approach for protein fold prediction from sequence. In *26th Annual International Conference on the IEEE EMBS*, pages 2885–2888, San Francisco.

Langlois, R. E., Diec, A., Perisic, O., Dai, Y., and Lu, H. (2006). Improved protein fold assignment using support vector machines. *International Journal of Bioinformatics Research and Applications*, **1**(3), 319–334.

Lee, S. and Grossmann., I. E. (2000). New algorithms for nonlinear generalized disjunctive programming. *Computers and Chemical Engineering Journal*, **24**(9-10), 2125–2141.

Lemmon, M. A. and Ferguson, K. M. (2000). Signal-dependent membrane targeting by Pleckstrin Homology (PH) domains. *Biochemical Journal*, **350**(1), 1–18.

Lemmon, M. A., Ferguson, K. M., and Abrams, C. S. (2002). Pleckstrin homology domains and the cytoskeleton. *FEBS Letters*, **513**(1), 71–76.

Leslie, C., Eskin, E., Weston, J., and Noble, W. S. (2002a). Mismatch string kernels for SVM protein classification. In *Advances in Neural Information Processing Systems*, volume 15, pages 1441–1448, Vancouver, British Columbia, Canada.

Leslie, C., Eskin, E., and Noble, W. S. (2002b). The spectrum kernel: A string kernel for SVM protein classification. In *Pacific Symposium on Biocomputing*, pages 564–575, Lihue, Hawaii.

Leslie, C., Eskin, E., Cohen, A., Weston, J., and Noble, W. S. (2004). Mismatch string kernels for discriminative protein classification. *Bioinformatics*, **20**(4), 467–476.

Letunic, I. and Bork, P. (2007). Interactive Tree Of Life (iTOL): An online tool for phylogenetic tree display and annotation. *Bioinformatics*, **23**(1), 127–128.

Li, L. (2001). Lemga: Learning models and generic algorithms. `http://www.work.caltech.edu/ling/lemga/`.

Li, W. and Godzik, A. (2006). CD-HIT: A fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics*, **22**(13), 1658–1659.

Liao, L. and Noble, W. S. (2002). Combining pairwise sequence similarity and support vector machines for remote protein homology detection. In *International Conference on Research in Computational Molecular Biology*, volume 6, pages 225–232, Washington DC, USA.

Lin, H. H., Han, L. Y., Zhang, H. L., Zheng, C. J., Xie, B., and Chen, Y. Z. (2006). Prediction of the functional class of lipid binding proteins from sequence-derived properties irrespective of sequence similarity. *Journal of Lipid Research*, **47**(4), 824–831.

Lin, H.-T., Lin, C.-J., and Weng, R. (2007). A note on platt's probabilistic outputs for support vector machines. *Machine Learning*, **68**(3), 267–276.

Liu, B., Zhao, K., Benkler, J., and Xiao, W. (2006). Rule interestingness analysis using OLAP operations. In *Special Interest Group on Knowledge Discovery and Data Mining*, volume 12, Philadelphia, USA. ACM.

Lodhi, H., Saunders, C., Shawe-Taylor, J., Cristianini, N., and Watkins, C. (2002). Text classification using string kernels. *Journal of Machine Learning Research*, **2**, 419–444.

Luscombe, N. M. and Thornton, J. M. (2002). Protein-DNA interactions: Amino acid conservation and the effects of mutations on binding specificity. *Journal of Molecular Biology*, **320**(5), 991–1009.

Luscombe, N. M., Austin, S. E., Berman, H. M., and Thornton, J. M. (2000). An overview of the structures of protein-DNA complexes. *Genome Biology*, **1**(Reviews001), 7558–7562.

MacKerell, A. D., Bashford, D., Bellott, M., Dunbrack, R. L., Evanseck, J. D., Field, M. J., Fischer, S., Gao, J., Guo, H., Ha, S., Joseph-McCarthy, D., Kuchnir, L., Kuczera, K., Lau, F. T. K., Mattos, C., Michnick, S., Ngo, T., Nguyen, D. T., Prodhom, B., Reiher, W. E., Roux, B., Schlenkrich, M., Smith, J. C., Stote, R., Straub, J., Watanabe, M., Wiorkiewicz-Kuczera, J., Yin, D., and Karplus, M. (1998). All-atom empirical potential for molecular modeling and dynamics studies of proteins. *Journal of Physical Chemistry B*, **102**(18), 3586–3616.

Malmberg, N. J., Buskirk, D. R. V., and Falke, J. J. (2003). Membrane-docking loops of the cPLA2 C2 domain: Detailed structural analysis of the protein-membrane interface via site-directed spin-labeling. *Biochemistry*, **42**(45), 13227–13240.

Manavalan, P. and Ponnuswamy, P. K. (1978). Hydrophobic character of amino acid residues in globular proteins. *Nature*, **275**(5681), 673–674.

Margineantu, D. D. and Dietterich, T. G. (1997). Pruning adaptive boosting. In *International Conference on Machine Learning*, volume 14, pages 211–218, Cavtat-Dubrovnik, Croatia. Morgan Kaufmann.

Maron, O. and Lozano-Pérez, T. (1998). A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*, volume 10, Denver, Colorado.

Mason, L., Baxter, J., Bartlett, P., and Frean, M. (1999). Boosting algorithms as gradient descent. In *Advances in Neural Information Processing Systems*, volume 12, pages 512–518, Denver, Colorado.

Mayer, K., Schuller, C., Wambutt, R., Murphy, G., Volckaert, G., Pohl, T., Dusterhoft, A., Stiekema, W., Entian, K. D., Terryn, N., Harris, B., Ansorge, W., Brandt, P., Grivell, L., Rieger, M., Weichselgartner, M., de Simone, V., Obermaier, B., Mache, R., Muller, M., Kreis, M., Delseny, M., Puigdomenech, P., Watson, M., Schmidtheini, T., Reichert, B., Portatelle, D., Pérez-Alonso, M., Boutry, M., Bancroft, I., Vos, P., Hoheisel, J., Zimmermann, W., Wedler, H., Ridley, P., Langham, S. A., McCullagh, B., Bilham, L., Robben, J., Van der Schueren, J., Grymonprez, B., Chuang, Y. J., Vandenbussche, F., Braeken, M., Weltjens, I., Voet, M., Bastiaens, I., Aert, R., Defoor, E., Weitzenegger, T., Bothe, G., Ramsperger, U., Hilbert, H., Braun, M., Holzer, E., Brandt, A., Peters, S., van Staveren, M., Dirkse, W., Mooijman, P., Lankhorst, R. K., Rose, M., Hauf, J., Kotter, P., Berneiser, S., Hempel, S., Feldpausch, M., Lamberth, S., Van den Daele, H., De Keyser, A., Buysshaert, C., Gielen, J., Villarroel, R., De Clercq, R., Van Montagu, M., Rogers, J., Cronin, A., Quail, M., Bray-Allen, S., Clark, L., Doggett, J., Hall, S., Kay, M., Lennard, N., McLay, K., Mayes, R., Pettett, A., Rajandream, M. A., Lyne, M., Benes, V., Rechmann, S., Borkova, D., Blocker, H., Scharfe, M., Grimm, M., Lohnert, T. H., Dose, S., de Haan, M., Maarse, A., Schafer, M., *et al.* (1999). Sequence and analysis of chromosome 4 of the plant Arabidopsis thaliana. *Nature*, **402**(6763), 769–777.

McGuffin, L. J. and Jones, D. T. (2003). Improvement of the GenTHREADER method for genomic fold recognition. *Proteins: Structure, Function, and Genetics*, **19**(7), 874–881.

McGuffin, L. J., Bryson, K., and Jones, D. T. (2000). The PSIPRED protein structure prediction server. *Bioinformatics*, **16**(4), 404–405.

McLaughlin, S. and Murray, D. (2005). Plasma membrane phosphoinositide organization by protein electrostatics. *Nature*, **438**(7068), 605–611.

McLaughlin, S., Wang, J., Gambhir, A., and Murray, D. (2002). PIP2 and proteins: Interactions, organization, and information flow. *Annual Review of Biophysics and Biomolecular Structure*, **31**(1), 151–175.

Melvin, I., Ie, E., Weston, J., Noble, W. S., and Leslie, C. (2007). Multi-class protein classification using adaptive codes. *Journal of Machine Learning Research*, **8**, 1557–1581.

Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). YALE: Rapid prototyping for complex data mining tasks. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, volume 12, Philadelphia, USA.

Mozsolits, H. and Aguilar, M.-I. (2002). Surface plasmon resonance spectroscopy: An emerging tool for the study of peptide-membrane interactions. *Peptide Science*, **66**(1), 3–18.

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**(4), 536–540.

Nastri, H. G., Evans, P. D., Walker, I. H., and Riggs, P. D. (1997). Catalytic and DNA binding properties of pvuII restriction endonuclease mutants. *Journal of Biological Chemistry*, **272**(41), 25761–25767.

Ng, P. C. and Henikoff, S. (2001). Predicting deleterious amino acid substitutions. *Genome Research*, **11**(5), 863–874.

Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, **5**(1), 71–99.

Pastinen, T., Ge, B., and Hudson, T. J. (2006). Influence of human genome polymorphism on gene expression. *Human Molecular Genetics*, **15**(Suppl 1), R9–16.

Pellegrini, M., Marcotte, E. M., Thompson, M. J., Eisenberg, D., and Yeates, T. O. (1999). Assigning protein functions by comparative genome analysis: Protein phylogenetic profiles. *Proceedings of the National Academy of Sciences*, **96**(8), 4285–4288.

Pellegrini-Calace, M. and Thornton, J. M. (2005). Detecting DNA-binding helix-turn-helix structural motifs using sequence and structure information. *Nucleic Acids Research*, **33**(7), 2129–2140.

Pingoud, A. and Jeltsch, A. (1997). Recognition and cleavage of DNA by type-II restriction endonucleases. *European Journal of Biochemistry*, **246**(1), 1–22.

Platt, J. C. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In P. J. Bartlett, B. Scholkopf, D. Schuurmans, and A. J. Smola, editors, *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, Boston.

Platt, J. C., Cristianini, N., and Shawe-Taylor, J. (1999). Large margin DAGs for multiclass classification. In *Advances in Neural Information Processing Systems*, volume 12, Denver, Colorado.

Ponnuswamy, P. K. (1993). Hydrophobic characteristics of folded proteins. *Progress in Biophysics and Molecular Biology*, **59**(1), 57–103.

Prabhakaran, M. (1990). The distribution of physical, chemical and conformational properties in signal and nascent peptides. *Biochemical Journal*, **269**(3), 691–696.

Pérez, M. S., Sanchez, A., Herrero, P., Robles, V., and Pena, J. M. (2005). Adapting the WEKA data mining toolkit to a grid based environment. In *Advances in Web Intelligence*, volume 3528, pages 492–497. Springer, Berlin / Heidelberg.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, pages 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers.

Quinlan, J. R. (1996). Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, **4**, 77–90.

Radzicka, A. and Wolfenden, R. (1988). Comparing the polarities of the amino acids: Side-chain distribution coefficients between the vapor phase, cyclohexane, 1-octanol, and neutral aqueous solution. *Biochemistry*, **27**(5), 1664–1670.

Rahmani, R. and Goldman, S. A. (2006). MISSL: Multiple-Instance Semi-Supervised Learning. In W. Cohen and A. Moore, editors, *International Conference on Machine Learning*, volume 23, Pittsburgh, USA. Omni Press.

Rakotomalala, R. (2005). TANAGRA : Un logiciel gratuit pour l'enseignement et la recherche. In *Actes de EGC*, volume 2, pages 697–702.

Rangwala, H. and Karypis, G. (2005). Profile-based direct kernels for remote homology detection and fold recognition. *Bioinformatics*, **21**(23), 4239–4247.

Ray, S. and Craven, M. (2005). Supervised versus multiple instance learning: An empirical comparison. In *International Conference on Machine Learning*, volume 22, pages 697–704, Bonn, Germany. ACM.

Ray, S. and Page, D. (2001). Multiple instance regression. In *International Conference on Machine Learning*, volume 18, pages 425–432.

Reich, D. E., Gabriel, S. B., and Altshuler, D. (2003). Quality and completeness of SNP databases. *Nature Genetics*, **33**(4), 457–458.

Ren, B., Robert, F., Wyrick, J. J., Aparicio, O., Jennings, E. G., Simon, I., Zeitlinger, J., Schreiber, J., Hannett, N., Kanin, E., Volkert, T. L., Wilson, C. J., Bell, S. P., and Young, R. A. (2000). Genome-wide location and function of DNA binding proteins. *Science*, **290**(5500), 2306–2309.

Riechmann, J. L., Heard, J., Martin, G., Reuber, L., Z, C., Jiang, Keddie, J., Adam, L., Pineda, O., Ratcliffe, O. J., Samaha, R. R., Creelman, R., Pilgrim, M., Broun, P., Zhang, J. Z., Ghandehari, D., Sherman, B. K., and L. Yu, G. (2000). Arabidopsis transcription factors: Genome-wide comparative analysis among eukaryotes. *Science*, **290**(5499), 2105–2110.

Rifkin, R. and Klautau, A. (2004). In defense of one-vs-all classifcation. *Journal of Machine Learning Research*, **5**, 101–141.

Rost, B. and Sander, C. (1993). Improved prediction of protein secondary structure by use of sequence profiles and neural networks. *Proceedings of the National Academy of Sciences*, **90**(16), 7558–7562.

Ruvkun, G. B. and Ausubel, F. M. (1981). A general method for site-directed mutagenesis in prokaryotes. *Nature*, **289**(5793), 85–88.

Saigo, H., Vert, J.-P., Ueda, N., and Akutsu, T. (2004). Protein homology detection using string alignment kernels. *Bioinformatics*, **20**(11), 1682–1689.

Saitou, N. and Nei, M. (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, **4**(4), 406–425.

Sanchez, R. and Sali, A. (1997). Evaluation of comparative protein structure modeling by MODELLER-3. *Proteins: Structure, Function, and Genetics*, **29**(S1), 50–58.

Sarai, A. and Kono, H. (2005). Protein-DNA recognition patterns and predictions. *Annual Review of Biophysics and Biomolecular Structure*, **34**(1), 379–398.

Schapire, R. E. and Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, **37**(3), 297–336.

Schneider, T. D., Stormo, G. D., Gold, L., and Ehrenfeucht, A. (1986). Information content of binding sites on nucleotide sequences. *Journal of Molecular Biology*, **188**(3), 415–431.

Shanahan, H. P., Garcia, M. A., Jones, S., and Thornton, J. M. (2004). Identifying DNA-binding proteins using structural motifs and the electrostatic potential. *Nucleic Acids Research*, **32**(16), 4732–4741.

Shen-Orr, S. S., Milo, R., Mangan, S., and Alon, U. (2002). Network motifs in the transcriptional regulation network of Escherichia coli. *Nature Genetics*, **31**(1), 64–68.

Sherry, S. T., Ward, M. H., Kholodov, M., Baker, J., Phan, L., Smigielski, E. M., and Sirotkin, K. (2001). dbSNP: The NCBI database of genetic variation. *Nucleic Acids Research*, **29**(1), 308–311.

Singh, S. M. and Murray, D. (2003). Molecular modeling of the membrane targeting of phospholipase C pleckstrin homology domains. *Protein Science*, **12**(9), 1934–1953.

Skolnick, J. and Kihara, D. (2001). Defrosting the frozen approximation: PROSPECTOR - a new approach to threading. *Proteins: Structure, Function, and Genetics*, **42**(3), 319–331.

Sonnenburg, S., Braun, M. L., Ong, C. S., Bengio, S., Bottou, L., Holmes, G., LeCun, Y., Muller, K.-R., Pereira, F., Rasmussen, C. E., Ratsch, G., Scholkopf, B., Smola, A., Vincent, P., Weston, J., and Williamson, R. (2007). The need for open source software in machine learning. *Journal of Machine Learning Research*, **8**, 2443–2466.

Stahelin, R. V., Long, F., Peter, B. J., Murray, D., De Camilli, P., McMahon, H. T., and Cho, W. (2003). Contrasting membrane interaction mechanisms of AP180 N-terminal homology (ANTH) and epsin N-terminal homology (ENTH) domains. *Journal of Biological Chemistry*, **278**(31), 28993–28999.

Stawiski, E. W., Gregoret, L. M., and Mandel-Gutfreund, Y. (2003). Annotating nucleic acid-binding function based on protein structure. *Journal of Molecular Biology*, **326**(4), 1065–1079.

Stormo, G. D., Schneider, T. D., Gold, L., and Ehrenfeucht, A. (1982). Use of the 'perceptron' algorithm to distinguish translational initiation sites in E. coli. *Nucleic Acids Research*, **10**(9), 2997–3011.

Stuart, J. M., Segal, E., Koller, D., and Kim, S. K. (2003). A gene-coexpression network for global discovery of conserved genetic modules. *Science*, **302**(5643), 249–255.

Suyama, M. and Ohara, O. (2003). DomCut: Prediction of inter-domain linker regions in amino acid sequences. *Bioinformatics*, **19**(5), 673–674.

Szilagyi, A. and Skolnick, J. (2006). Efficient prediction of nucleic acid binding function from low-resolution protein structures. *Journal of Molecular Biology*, **358**(3), 922–933.

Teruel, M. N. and Meyer, T. (2000). Translocation and reversible localization of signaling proteins: A dynamic future for signal transduction. *Cell*, **103**(2), 181–184.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, New York.

Vassylyev, D. G., Kashiwagi, T., Mikami, Y., Ariyoshi, M., Iwai, S., Ohtsuka, E., and Morikawa, K. (1995). Atomic model of a pyrimidine dimer excision repair enzyme complexed with a DNA substrate: Structural basis for damaged DNA recognition. *Cell*, **83**(5), 773–782.

Viola, P., Platt, J. C., and Zhang, C. (2006). Multiple instance boosting for object detection. In Y. Weiss, B. Scholkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems*, volume 18. MIT Press.

Wang, G. and Dunbrack, Roland L., J. (2003). PISCES: a protein sequence culling server. *Bioinformatics*, **19**(12), 1589–1591.

Wang, J. and Zucker, J.-D. (2000). Solving the multiple-instance problem: A lazy learning approach. In *International Conference on Machine Learning*, volume 17, pages 1119–1125.

Wei, L. and Altman, R. B. (2003). Recognizing complex, asymmetric functional sites in protein structures using a Bayesian scoring function. *Journal of Bioinformatics and Computational Biology*, **1**(1), 119–138.

Weston, J., Elisseeff, A., BakIr, G., and Sinz, F. (2003). Machine learning library for Matlab. `http://www.kyb.tuebingen.mpg.de/bs/people/spider/main.html`.

Weston, J., Leslie, C., Ie, E., Zhou, D., Elisseeff, A., and Noble, W. S. (2005). Semi-supervised protein classification using cluster kernels. *Bioinformatics*, **21**(15), 3241–3247.

White, A., Ding, X., vanderSpek, J. C., Murphy, J. R., and Ringe, D. (1998). Structure of the metal-ion-activated diphtheria toxin repressor/tox operator complex. *Nature*, **394**(6692), 502–506.

Witten, I. H. and Frank, E. (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, San Francisco, 2nd edition.

Wolpert, D. H. (2001). The supervised learning no-free-lunch theorems. In *Online World Conference on Soft Computing in Industrial Applications*, volume 6, WWW.

Wolpert, D. H. and Macready, W. G. (1995). No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Wu, P. G. and Brand, L. (1994). Resonance energy transfer: Methods and applications. *Analytical Biochemistry*, **218**(1), 1–13.

Wuthrich, K. (1990). Protein structure determination in solution by NMR spectroscopy. *Journal of Biological Chemistry*, **265**(36), 22059–22062.

Xu, X. and Frank, E. (2004). Logistic regression and boosting for labeled bags of instances. In *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 272–281. Springer, Heidelberg.

Xu, Y. and Xu, D. (2000). Protein threading using PROSPECT: Design and evaluation. *Proteins: Structure, Function, and Genetics*, **40**(3), 343–354.

Yan, C., Terribilini, M., Wu, F., Jernigan, R., Dobbs, D., and Honavar, V. (2006). Predicting DNA-binding sites of proteins from amino acid sequence. *BMC Bioinformatics*, **7**(1), 262–272.

Ye, Z.-Q., Zhao, S.-Q., Gao, G., Liu, X.-Q., Langlois, R. E., Lu, H., and Wei, L. (2007). Finding new structural and sequence attributes to predict possible disease association of Single Amino acid Polymorphism (SAP). *Bioinformatics*, **23**(12), 1444–1450.

Yip, Y. L., Scheib, H., Diemand, A. V., Gattiker, A., Famiglietti, L. M., Gasteiger, E., and Bairoch, A. (2004). The Swiss-Prot variant page and the ModSNP database: A resource for sequence and structure information on human protein variants. *Human Mutation*, **23**(5), 464–470.

Yu, C.-S., Wang, J.-Y., Yang, J.-M., Lyu, P.-C., Lin, C.-J., and Hwang, J.-K. (2003). Fine-grained protein fold assignment by support vector machines using generalized n-peptide coding schemes and jury voting from multiple-parameter sets. *Proteins: Structure, Function, and Genetics*, **50**(4), 531–536.

Yu, H., Luscombe, N. M., Lu, H. X., Zhu, X., Xia, Y., Han, J.-D. J., Bertin, N., Chung, S., Vidal, M., and Gerstein, M. (2004). Annotation transfer between genomes: Protein-protein interologs and protein-DNA regulogs. *Genome Research*, **14**(6), 1107–1118.

Zadrozny, B. and Elkan, C. (2002). Transforming classifier scores into accurate multiclass probability estimates. In *Special Interest Group on Knowledge Discovery and Data Mining*, volume 8, pages 694–699, Edmonton, Alberta, Canada. ACM Press.

Zadrozny, B., Langford, J., and Abe, N. (2003). Cost-sensitive learning by cost-proportionate example weighting. In *IEEE International Conference on Data Mining*, volume 3, page 435, Melbourne, Florida.

Zhang, Q. and Goldman, S. A. (2001). EM-DD: An improved multiple-instance learning technique. In *Neural Information Processing Systems*, volume 14, Vancouver, British Columbia, Canada.

Zhang, Q., Goldman, S. A., Yu, W., and Fritts, J. (2002). Content-based image retrieval using multiple-instance learning. In *International Conference on Machine Learning*, pages 682–689, Sydney, Australia. Morgan Kaufmann Publishers Inc.

Zhou, H. and Zhou, Y. (2004). Quantifying the effect of burial of amino acid residues on protein stability. *Bioinformatics*, **54**(2), 315–322.

Zimmerman, J. M., Eliezer, N., and Simha, R. (1968). The characterization of amino acid sequences in proteins by statistical methods. *Journal of Theoretical Biology*, **21**(2), 170–201.

# APPENDICES

Table A.1: SwissProt DNA-binding Protein Dataset

| Dataset | Tot | DNA | Non |
|---|---|---|---|
| C. elegan | 3099 | 284 | 942 |
| Drosophila | 2689 | 347 | 998 |
| A. thaliana | 6207 | 731 | 1874 |
| E. coli | 4321 | 493 | 1855 |
| Saccharomyces | 6556 | 568 | 2672 |
| Mus musculus | 14273 | 1275 | 6267 |
| Homo sapiens | 17658 | 1898 | 7498 |
| All | 289473 | 20270 | 112393 |

[a] Tot represents the total number in SwissProt not the sum of Non and DNA

# Appendix A

# Dataset Statistics for Chapter 7

A.1 summarizes the statistics of the unfiltered genome-wide datasets derived from the SwissProt. This table comprises data from the seven organisms and the comprehensive dataset; the first column (Tot) holds the total number of sequences in SwissProt. The second and third columns tally the number of sequences in each of the selected subsets. Note, the second and third columns do not add to the first column since they are tabulated from different sources.

A.2 summarizes the statistics of the filtered genome-wide datasets at 40% and 25% pairwise sequence identity. This table has two sets of three columns tabulating the number of total DNA-binding, and non-binding protein sequence examples in each of the seven organism datasets as well as the comprehensive dataset.

Table A.2: Filtered DNA-binding Protein Datasets

| Dataset | 25% Identity | | | 40% Identity | | |
|---|---|---|---|---|---|---|
| | Tot | DNA | Non | Tot | DNA | Non |
| C. elegan | 469 | 115 | 354 | 825 | 216 | 609 |
| Drosophila | 596 | 207 | 389 | 923 | 270 | 653 |
| A. thaliana | 487 | 161 | 326 | 1007 | 399 | 608 |
| E. coli | 559 | 130 | 429 | 808 | 207 | 601 |
| Saccharomyces | 1945 | 365 | 1580 | 2464 | 464 | 2000 |
| Mus musculus | 749 | 136 | 613 | 1205 | 232 | 973 |
| Homo sapiens | 2267 | 483 | 1784 | 3450 | 787 | 2663 |
| All | 15611 | 3053 | 12558 | 28236 | 5634 | 22602 |

# Appendix B

# Proteome Analysis for Chapter 7

The following table compares blast and boosted trees using 10-fold cross-validation over the same set of partitions. This is measured using four metrics: accuracy (Acc.), sensitivity (Sen.), specificity (Spe.) and area under the receiver operating characteristic curve (AUR). Each dataset has been filtered such that no two proteins share more than 25% and 40 % sequence identity *e.g.* no protein in the C. elegan dataset has more than 25% and 40 % sequence identity with any protein in each training set organism. This data was used to create the plots in Chapter 7.

Table B.1: Ten-fold cross-validation Proteome Comparison

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| All | Boosted Trees | 85.30 | 46.19 | 94.81 | 88.71 | 86.21 | 51.34 | 94.90 | 89.78 |
| | Blast | 88.08 | 73.73 | 91.57 | 86.45 | 94.18 | 86.60 | 96.08 | 94.64 |
| C. elegan | Boosted Trees | 89.98 | 73.90 | 95.20 | 93.94 | 90.66 | 78.93 | 94.92 | 95.89 |
| | Blast | 84.65 | 73.91 | 88.14 | 86.06 | 88.73 | 81.02 | 91.46 | 93.81 |
| Drosophila | Boosted Trees | 83.22 | 72.97 | 88.68 | 90.40 | 87.10 | 74.12 | 92.48 | 93.05 |
| | Blast | 81.21 | 74.40 | 84.83 | 88.26 | 85.81 | 78.15 | 88.97 | 91.63 |
| E. coli | Boosted Trees | 88.02 | 68.92 | 93.75 | 92.29 | 87.87 | 73.38 | 92.84 | 93.28 |
| | Blast | 84.08 | 61.54 | 90.91 | 85.23 | 88.99 | 78.74 | 92.51 | 92.27 |
| Earcress | Boosted Trees | 87.08 | 75.81 | 92.67 | 92.14 | 90.57 | 88.73 | 91.78 | 96.64 |
| | Blast | 85.01 | 83.85 | 85.58 | 91.01 | 90.67 | 94.99 | 87.83 | 97.84 |
| Human | Boosted Trees | 85.00 | 45.37 | 95.74 | 87.55 | 86.61 | 57.30 | 95.27 | 90.57 |
| | Blast | 82.05 | 67.70 | 85.93 | 81.30 | 87.48 | 77.89 | 90.31 | 89.58 |
| Mouse | Boosted Trees | 85.19 | 37.62 | 95.76 | 83.49 | 86.64 | 47.02 | 96.10 | 87.48 |
| | Blast | 81.98 | 59.56 | 86.95 | 77.78 | 87.30 | 71.98 | 90.96 | 86.76 |
| Yeast | Boosted Trees | 83.80 | 34.25 | 95.25 | 83.82 | 84.74 | 37.93 | 95.60 | 85.96 |
| | Blast | 76.50 | 53.97 | 81.71 | 70.62 | 80.88 | 62.28 | 85.20 | 79.55 |

Table C.1: Train on each organism and test on C. elegan

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| Drosophila | Boosted Trees | 86.35 | 58.26 | 95.48 | 93.03 | 86.06 | 56.94 | 96.39 | 93.97 |
| | Blast | 84.86 | 89.57 | 83.33 | 93.94 | 91.03 | 93.06 | 90.31 | 97.00 |
| E. coli | Boosted Trees | 78.68 | 49.57 | 88.14 | 86.75 | 79.15 | 54.17 | 88.01 | 86.42 |
| | Blast | 71.86 | 22.61 | 87.85 | 66.26 | 72.00 | 31.48 | 86.37 | 67.68 |
| Earcress | Boosted Trees | 84.01 | 57.39 | 92.66 | 88.86 | 83.52 | 60.19 | 91.79 | 90.22 |
| | Blast | 71.00 | 55.65 | 75.99 | 69.03 | 72.61 | 58.80 | 77.50 | 74.40 |
| Human | Boosted Trees | 82.73 | 33.91 | 98.59 | 92.70 | 81.33 | 32.41 | 98.69 | 93.92 |
| | Blast | 94.67 | 90.43 | 96.05 | 95.26 | 92.97 | 88.43 | 94.58 | 94.07 |
| Mouse | Boosted Trees | 78.89 | 13.91 | 100.00 | 91.05 | 77.94 | 15.74 | 100.00 | 91.84 |
| | Blast | 89.13 | 80.87 | 91.81 | 89.10 | 91.88 | 86.57 | 93.76 | 93.18 |
| Yeast | Boosted Trees | 79.10 | 22.61 | 97.46 | 88.04 | 78.06 | 25.93 | 96.55 | 87.68 |
| | Blast | 80.38 | 57.39 | 87.85 | 77.36 | 79.64 | 56.48 | 87.85 | 76.63 |

# Appendix C

# Test on One Organism for Chapter 7

The following tables compare boosted trees and blast trained on each organism in the first column and tested on a single left out organism in the caption. This is measured using four metrics: accuracy (Acc.), sensitivity (Sen.), specificity (Spe.) and area under the receiver operating characteristic curve (AUR). Each dataset has been filtered such that no two proteins share more than 25% and 40 % sequence identity *e.g.* no protein in the C. elegan dataset has more than 25% and 40 % sequence identity with any protein in each training set organism. This data was used to create the plots in Chapter 7.

Table C.2: Train on each organism and test on Drosophila

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 84.06 | 71.50 | 90.75 | 91.23 | 86.02 | 67.04 | 93.87 | 92.76 |
| | Blast | 81.38 | 71.01 | 86.89 | 86.40 | 87.22 | 76.67 | 91.58 | 92.39 |
| E. coli | Boosted Trees | 67.62 | 34.30 | 85.35 | 73.51 | 68.91 | 34.07 | 83.31 | 75.20 |
| | Blast | 61.91 | 18.84 | 84.83 | 65.59 | 66.20 | 22.22 | 84.38 | 63.97 |
| Earcress | Boosted Trees | 79.19 | 81.64 | 77.89 | 88.16 | 83.10 | 82.22 | 83.46 | 91.05 |
| | Blast | 67.28 | 63.29 | 69.41 | 71.77 | 73.89 | 69.63 | 75.65 | 78.15 |
| Human | Boosted Trees | 81.88 | 60.87 | 93.06 | 90.33 | 85.37 | 62.59 | 94.79 | 93.48 |
| | Blast | 93.79 | 90.34 | 95.63 | 95.27 | 90.68 | 86.67 | 92.34 | 93.81 |
| Mouse | Boosted Trees | 80.20 | 50.24 | 96.14 | 90.83 | 85.81 | 58.52 | 97.09 | 92.98 |
| | Blast | 84.90 | 75.85 | 89.72 | 87.57 | 86.89 | 77.78 | 90.66 | 90.69 |
| Yeast | Boosted Trees | 72.48 | 41.06 | 89.20 | 84.23 | 77.36 | 40.37 | 92.65 | 87.03 |
| | Blast | 78.69 | 64.73 | 86.12 | 83.99 | 79.85 | 68.52 | 84.53 | 84.47 |

Table C.3: Train on each organism and test on E. coli

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 77.46 | 10.00 | 97.90 | 77.49 | 75.74 | 8.21 | 99.00 | 83.68 |
| | Blast | 71.74 | 23.85 | 86.25 | 59.01 | 71.78 | 24.64 | 88.02 | 59.66 |
| Drosophila | Boosted Trees | 76.74 | 3.08 | 99.07 | 78.60 | 74.75 | 2.42 | 99.67 | 80.99 |
| | Blast | 71.74 | 25.38 | 85.78 | 56.97 | 70.92 | 26.09 | 86.36 | 58.80 |
| Earcress | Boosted Trees | 78.00 | 10.00 | 98.60 | 80.73 | 75.25 | 6.76 | 98.84 | 81.10 |
| | Blast | 66.73 | 30.00 | 77.86 | 56.45 | 65.97 | 30.43 | 78.20 | 60.28 |
| Human | Boosted Trees | 77.46 | 3.85 | 99.77 | 80.19 | 74.88 | 2.42 | 99.83 | 83.75 |
| | Blast | 71.91 | 16.92 | 88.58 | 54.02 | 69.80 | 18.84 | 87.35 | 55.23 |
| Mouse | Boosted Trees | 76.74 | 0.77 | 99.77 | 78.30 | 74.26 | 0.00 | 99.83 | 85.66 |
| | Blast | 71.91 | 18.46 | 88.11 | 55.96 | 71.66 | 19.32 | 89.68 | 56.94 |
| Yeast | Boosted Trees | 77.28 | 3.08 | 99.77 | 82.02 | 74.75 | 2.42 | 99.67 | 82.45 |
| | Blast | 69.95 | 28.46 | 82.52 | 60.77 | 71.29 | 31.88 | 84.86 | 64.22 |

Table C.4: Train on each organism and test on Earcress

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 82.75 | 62.11 | 92.94 | 89.57 | 83.71 | 66.17 | 95.23 | 94.03 |
| | Blast | 68.58 | 42.24 | 81.60 | 69.14 | 69.81 | 45.86 | 85.53 | 76.53 |
| Drosophila | Boosted Trees | 87.06 | 69.57 | 95.71 | 92.19 | 86.20 | 69.17 | 97.37 | 94.87 |
| | Blast | 68.38 | 60.25 | 72.39 | 66.27 | 70.80 | 58.90 | 78.62 | 74.59 |
| E. coli | Boosted Trees | 71.25 | 44.72 | 84.36 | 81.30 | 72.00 | 47.12 | 88.32 | 83.68 |
| | Blast | 63.45 | 13.04 | 88.34 | 56.98 | 59.09 | 16.79 | 86.84 | 60.81 |
| Human | Boosted Trees | 76.59 | 33.54 | 97.85 | 87.02 | 71.40 | 30.58 | 98.19 | 90.46 |
| | Blast | 79.26 | 55.90 | 90.80 | 83.81 | 75.07 | 52.63 | 89.80 | 82.24 |
| Mouse | Boosted Trees | 72.28 | 16.77 | 99.69 | 88.24 | 67.33 | 18.30 | 99.51 | 91.76 |
| | Blast | 71.46 | 41.61 | 86.20 | 71.51 | 71.60 | 43.11 | 90.30 | 78.14 |
| Yeast | Boosted Trees | 71.66 | 17.39 | 98.47 | 88.76 | 68.12 | 22.06 | 98.36 | 91.35 |
| | Blast | 75.98 | 49.69 | 88.96 | 77.76 | 73.09 | 48.62 | 89.14 | 80.50 |

Table C.5: Train on each organism and test on Human

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 79.71 | 57.56 | 85.71 | 82.31 | 82.78 | 60.48 | 89.37 | 86.78 |
| | Blast | 78.30 | 57.56 | 83.91 | 74.33 | 82.93 | 71.16 | 86.41 | 85.01 |
| Drosophila | Boosted Trees | 80.46 | 55.90 | 87.11 | 82.57 | 83.39 | 61.50 | 89.86 | 87.45 |
| | Blast | 73.40 | 61.08 | 76.74 | 74.02 | 81.16 | 73.70 | 83.36 | 85.08 |
| E. coli | Boosted Trees | 74.11 | 51.14 | 80.33 | 76.41 | 74.49 | 57.31 | 79.57 | 79.71 |
| | Blast | 70.45 | 21.33 | 83.74 | 55.98 | 68.96 | 24.40 | 82.13 | 58.57 |
| Earcress | Boosted Trees | 72.87 | 67.29 | 74.38 | 79.29 | 76.72 | 71.16 | 78.37 | 83.72 |
| | Blast | 66.96 | 54.87 | 70.24 | 63.19 | 69.16 | 62.26 | 71.20 | 70.88 |
| Mouse | Boosted Trees | 82.71 | 31.26 | 96.64 | 82.70 | 85.22 | 48.92 | 95.94 | 88.22 |
| | Blast | 80.99 | 56.94 | 87.50 | 75.65 | 85.86 | 69.38 | 90.72 | 84.95 |
| Yeast | Boosted Trees | 78.65 | 28.16 | 92.32 | 78.01 | 78.70 | 27.06 | 93.95 | 80.61 |
| | Blast | 81.74 | 62.53 | 86.94 | 76.11 | 82.70 | 69.89 | 86.48 | 81.82 |

Table C.6: Train on each organism and test on Mouse

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 80.37 | 62.50 | 84.34 | 84.14 | 83.82 | 68.10 | 87.56 | 86.76 |
| | Blast | 82.11 | 65.44 | 85.81 | 80.81 | 84.65 | 71.55 | 87.77 | 86.37 |
| Drosophila | Boosted Trees | 81.31 | 63.24 | 85.32 | 84.69 | 84.56 | 65.95 | 89.00 | 87.97 |
| | Blast | 76.90 | 66.18 | 79.28 | 80.86 | 84.07 | 77.16 | 85.71 | 88.24 |
| E. coli | Boosted Trees | 73.83 | 47.79 | 79.61 | 75.21 | 74.44 | 52.16 | 79.75 | 78.48 |
| | Blast | 74.90 | 25.74 | 85.81 | 61.87 | 72.28 | 24.57 | 83.66 | 62.15 |
| Earcress | Boosted Trees | 74.77 | 74.26 | 74.88 | 80.91 | 76.68 | 72.84 | 77.60 | 84.20 |
| | Blast | 67.29 | 58.09 | 69.33 | 64.97 | 70.71 | 67.24 | 71.53 | 74.40 |
| Human | Boosted Trees | 85.05 | 50.00 | 92.82 | 84.63 | 86.56 | 59.05 | 93.11 | 90.00 |
| | Blast | 95.99 | 88.97 | 97.55 | 94.72 | 89.21 | 84.91 | 90.24 | 92.03 |
| Yeast | Boosted Trees | 80.24 | 25.00 | 92.50 | 76.56 | 80.08 | 22.41 | 93.83 | 78.13 |
| | Blast | 79.71 | 58.82 | 84.34 | 74.13 | 80.91 | 64.66 | 84.79 | 80.34 |

Table C.7: Train on each organism and test on Yeast

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 79.74 | 41.37 | 88.61 | 78.55 | 79.87 | 37.93 | 89.60 | 79.17 |
| | Blast | 72.19 | 33.70 | 81.08 | 56.55 | 74.31 | 45.69 | 80.95 | 67.03 |
| Drosophila | Boosted Trees | 80.05 | 47.95 | 87.47 | 80.73 | 80.64 | 48.28 | 88.15 | 82.31 |
| | Blast | 70.33 | 46.85 | 75.76 | 60.07 | 74.23 | 53.45 | 79.05 | 67.63 |
| E. coli | Boosted Trees | 71.77 | 48.49 | 77.15 | 72.16 | 72.04 | 50.65 | 77.00 | 73.42 |
| | Blast | 73.78 | 23.84 | 85.32 | 56.70 | 73.09 | 25.00 | 84.25 | 58.40 |
| Earcress | Boosted Trees | 76.66 | 51.23 | 82.53 | 78.49 | 76.91 | 56.03 | 81.75 | 79.76 |
| | Blast | 65.09 | 54.25 | 67.59 | 61.67 | 66.84 | 56.47 | 69.25 | 66.39 |
| Human | Boosted Trees | 81.44 | 20.82 | 95.44 | 74.41 | 81.86 | 23.28 | 95.45 | 77.86 |
| | Blast | 80.15 | 52.33 | 86.58 | 73.52 | 78.73 | 52.80 | 84.75 | 73.38 |
| Mouse | Boosted Trees | 81.13 | 6.30 | 98.42 | 72.06 | 81.33 | 8.19 | 98.30 | 74.78 |
| | Blast | 73.52 | 32.88 | 82.91 | 55.52 | 76.06 | 41.16 | 84.15 | 64.18 |

Table D.1: Train on C. elegan and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| Drosophila | Boosted Trees | 84.06 | 71.50 | 90.75 | 91.23 | 86.02 | 67.04 | 93.87 | 92.76 |
| | Blast | 81.38 | 71.01 | 86.89 | 86.40 | 87.22 | 76.67 | 91.58 | 92.39 |
| E. coli | Boosted Trees | 77.46 | 10.00 | 97.90 | 77.49 | 75.74 | 8.21 | 99.00 | 83.68 |
| | Blast | 71.74 | 23.85 | 86.25 | 59.01 | 71.78 | 24.64 | 88.02 | 59.66 |
| Earcress | Boosted Trees | 82.75 | 62.11 | 92.94 | 89.57 | 83.71 | 66.17 | 95.23 | 94.03 |
| | Blast | 68.58 | 42.24 | 81.60 | 69.14 | 69.81 | 45.86 | 85.53 | 76.53 |
| Human | Boosted Trees | 79.71 | 57.56 | 85.71 | 82.31 | 82.78 | 60.48 | 89.37 | 86.78 |
| | Blast | 78.30 | 57.56 | 83.91 | 74.33 | 82.93 | 71.16 | 86.41 | 85.01 |
| Mouse | Boosted Trees | 80.37 | 62.50 | 84.34 | 84.14 | 83.82 | 68.10 | 87.56 | 86.76 |
| | Blast | 82.11 | 65.44 | 85.81 | 80.81 | 84.65 | 71.55 | 87.77 | 86.37 |
| Yeast | Boosted Trees | 79.74 | 41.37 | 88.61 | 78.55 | 79.87 | 37.93 | 89.60 | 79.17 |
| | Blast | 72.19 | 33.70 | 81.08 | 56.55 | 74.31 | 45.69 | 80.95 | 67.03 |

# Appendix D

# Test on One Organism for Chapter 7

The following tables compare boosted trees and blast trained on one organism and tested on each organism in the first column. This is measured using four metrics: accuracy (Acc.), sensitivity (Sen.), specificity (Spe.) and area under the receiver operating characteristic curve (AUR). Each dataset has been filtered such that no two proteins share more than 25% and 40 % sequence identity *e.g.* no protein in the C. elegan dataset has more than 25% and 40 % sequence identity with any protein in each training set organism. This data was used to create the plots in Chapter 7.

Table D.2: Train on Drosophila and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 86.35 | 58.26 | 95.48 | 93.03 | 86.06 | 56.94 | 96.39 | 93.97 |
| | Blast | 84.86 | 89.57 | 83.33 | 93.94 | 91.03 | 93.06 | 90.31 | 97.00 |
| E. coli | Boosted Trees | 76.74 | 3.08 | 99.07 | 78.60 | 74.75 | 2.42 | 99.67 | 80.99 |
| | Blast | 71.74 | 25.38 | 85.78 | 56.97 | 70.92 | 26.09 | 86.36 | 58.80 |
| Earcress | Boosted Trees | 87.06 | 69.57 | 95.71 | 92.19 | 86.20 | 69.17 | 97.37 | 94.87 |
| | Blast | 68.38 | 60.25 | 72.39 | 66.27 | 70.80 | 58.90 | 78.62 | 74.59 |
| Human | Boosted Trees | 80.46 | 55.90 | 87.11 | 82.57 | 83.39 | 61.50 | 89.86 | 87.45 |
| | Blast | 73.40 | 61.08 | 76.74 | 74.02 | 81.16 | 73.70 | 83.36 | 85.08 |
| Mouse | Boosted Trees | 81.31 | 63.24 | 85.32 | 84.69 | 84.56 | 65.95 | 89.00 | 87.97 |
| | Blast | 76.90 | 66.18 | 79.28 | 80.86 | 84.07 | 77.16 | 85.71 | 88.24 |
| Yeast | Boosted Trees | 80.05 | 47.95 | 87.47 | 80.73 | 80.64 | 48.28 | 88.15 | 82.31 |
| | Blast | 70.33 | 46.85 | 75.76 | 60.07 | 74.23 | 53.45 | 79.05 | 67.63 |

Table D.3: Train on E. coli and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 78.68 | 49.57 | 88.14 | 86.75 | 79.15 | 54.17 | 88.01 | 86.42 |
| | Blast | 71.86 | 22.61 | 87.85 | 66.26 | 72.00 | 31.48 | 86.37 | 67.68 |
| Drosophila | Boosted Trees | 67.62 | 34.30 | 85.35 | 73.51 | 68.91 | 34.07 | 83.31 | 75.20 |
| | Blast | 61.91 | 18.84 | 84.83 | 65.59 | 66.20 | 22.22 | 84.38 | 63.97 |
| Earcress | Boosted Trees | 71.25 | 44.72 | 84.36 | 81.30 | 72.00 | 47.12 | 88.32 | 83.68 |
| | Blast | 63.45 | 13.04 | 88.34 | 56.98 | 59.09 | 16.79 | 86.84 | 60.81 |
| Human | Boosted Trees | 74.11 | 51.14 | 80.33 | 76.41 | 74.49 | 57.31 | 79.57 | 79.71 |
| | Blast | 70.45 | 21.33 | 83.74 | 55.98 | 68.96 | 24.40 | 82.13 | 58.57 |
| Mouse | Boosted Trees | 73.83 | 47.79 | 79.61 | 75.21 | 74.44 | 52.16 | 79.75 | 78.48 |
| | Blast | 74.90 | 25.74 | 85.81 | 61.87 | 72.28 | 24.57 | 83.66 | 62.15 |
| Yeast | Boosted Trees | 71.77 | 48.49 | 77.15 | 72.16 | 72.04 | 50.65 | 77.00 | 73.42 |
| | Blast | 73.78 | 23.84 | 85.32 | 56.70 | 73.09 | 25.00 | 84.25 | 58.40 |

Table D.4: Train on Earcress and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 84.01 | 57.39 | 92.66 | 88.86 | 83.52 | 60.19 | 91.79 | 90.22 |
| | Blast | 71.00 | 55.65 | 75.99 | 69.03 | 72.61 | 58.80 | 77.50 | 74.40 |
| Drosophila | Boosted Trees | 79.19 | 81.64 | 77.89 | 88.16 | 83.10 | 82.22 | 83.46 | 91.05 |
| | Blast | 67.28 | 63.29 | 69.41 | 71.77 | 73.89 | 69.63 | 75.65 | 78.15 |
| E. coli | Boosted Trees | 78.00 | 10.00 | 98.60 | 80.73 | 75.25 | 6.76 | 98.84 | 81.10 |
| | Blast | 66.73 | 30.00 | 77.86 | 56.45 | 65.97 | 30.43 | 78.20 | 60.28 |
| Human | Boosted Trees | 72.87 | 67.29 | 74.38 | 79.29 | 76.72 | 71.16 | 78.37 | 83.72 |
| | Blast | 66.96 | 54.87 | 70.24 | 63.19 | 69.16 | 62.26 | 71.20 | 70.88 |
| Mouse | Boosted Trees | 74.77 | 74.26 | 74.88 | 80.91 | 76.68 | 72.84 | 77.60 | 84.20 |
| | Blast | 67.29 | 58.09 | 69.33 | 64.97 | 70.71 | 67.24 | 71.53 | 74.40 |
| Yeast | Boosted Trees | 76.66 | 51.23 | 82.53 | 78.49 | 76.91 | 56.03 | 81.75 | 79.76 |
| | Blast | 65.09 | 54.25 | 67.59 | 61.67 | 66.84 | 56.47 | 69.25 | 66.39 |

Table D.5: Train on Human and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 82.73 | 33.91 | 98.59 | 92.70 | 81.33 | 32.41 | 98.69 | 93.92 |
| | Blast | 94.67 | 90.43 | 96.05 | 95.26 | 92.97 | 88.43 | 94.58 | 94.07 |
| Drosophila | Boosted Trees | 81.88 | 60.87 | 93.06 | 90.33 | 85.37 | 62.59 | 94.79 | 93.48 |
| | Blast | 93.79 | 90.34 | 95.63 | 95.27 | 90.68 | 86.67 | 92.34 | 93.81 |
| E. coli | Boosted Trees | 77.46 | 3.85 | 99.77 | 80.19 | 74.88 | 2.42 | 99.83 | 83.75 |
| | Blast | 71.91 | 16.92 | 88.58 | 54.02 | 69.80 | 18.84 | 87.35 | 55.23 |
| Earcress | Boosted Trees | 76.59 | 33.54 | 97.85 | 87.02 | 71.40 | 30.58 | 98.19 | 90.46 |
| | Blast | 79.26 | 55.90 | 90.80 | 83.81 | 75.07 | 52.63 | 89.80 | 82.24 |
| Mouse | Boosted Trees | 85.05 | 50.00 | 92.82 | 84.63 | 86.56 | 59.05 | 93.11 | 90.00 |
| | Blast | 95.99 | 88.97 | 97.55 | 94.72 | 89.21 | 84.91 | 90.24 | 92.03 |
| Yeast | Boosted Trees | 81.44 | 20.82 | 95.44 | 74.41 | 81.86 | 23.28 | 95.45 | 77.86 |
| | Blast | 80.15 | 52.33 | 86.58 | 73.52 | 78.73 | 52.80 | 84.75 | 73.38 |

Table D.6: Train on Mouse and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 78.89 | 13.91 | 100.00 | 91.05 | 77.94 | 15.74 | 100.00 | 91.84 |
| | Blast | 89.13 | 80.87 | 91.81 | 89.10 | 91.88 | 86.57 | 93.76 | 93.18 |
| Drosophila | Boosted Trees | 80.20 | 50.24 | 96.14 | 90.83 | 85.81 | 58.52 | 97.09 | 92.98 |
| | Blast | 84.90 | 75.85 | 89.72 | 87.57 | 86.89 | 77.78 | 90.66 | 90.69 |
| E. coli | Boosted Trees | 76.74 | 0.77 | 99.77 | 78.30 | 74.26 | 0.00 | 99.83 | 85.66 |
| | Blast | 71.91 | 18.46 | 88.11 | 55.96 | 71.66 | 19.32 | 89.68 | 56.94 |
| Earcress | Boosted Trees | 72.28 | 16.77 | 99.69 | 88.24 | 67.33 | 18.30 | 99.51 | 91.76 |
| | Blast | 71.46 | 41.61 | 86.20 | 71.51 | 71.60 | 43.11 | 90.30 | 78.14 |
| Human | Boosted Trees | 82.71 | 31.26 | 96.64 | 82.70 | 85.22 | 48.92 | 95.94 | 88.22 |
| | Blast | 80.99 | 56.94 | 87.50 | 75.65 | 85.86 | 69.38 | 90.72 | 84.95 |
| Yeast | Boosted Trees | 81.13 | 6.30 | 98.42 | 72.06 | 81.33 | 8.19 | 98.30 | 74.78 |
| | Blast | 73.52 | 32.88 | 82.91 | 55.52 | 76.06 | 41.16 | 84.15 | 64.18 |

Table D.7: Train on Yeast and test on each organism

| | | 25% Identity | | | | 40% Identity | | | |
|---|---|---|---|---|---|---|---|---|---|
| Organism | Algorithm | Acc. | Sen. | Spe. | AUR | Acc. | Sen. | Spe. | AUR |
| C. elegan | Boosted Trees | 79.10 | 22.61 | 97.46 | 88.04 | 78.06 | 25.93 | 96.55 | 87.68 |
| | Blast | 80.38 | 57.39 | 87.85 | 77.36 | 79.64 | 56.48 | 87.85 | 76.63 |
| Drosophila | Boosted Trees | 72.48 | 41.06 | 89.20 | 84.23 | 77.36 | 40.37 | 92.65 | 87.03 |
| | Blast | 78.69 | 64.73 | 86.12 | 83.99 | 79.85 | 68.52 | 84.53 | 84.47 |
| E. coli | Boosted Trees | 77.28 | 3.08 | 99.77 | 82.02 | 74.75 | 2.42 | 99.67 | 82.45 |
| | Blast | 69.95 | 28.46 | 82.52 | 60.77 | 71.29 | 31.88 | 84.86 | 64.22 |
| Earcress | Boosted Trees | 71.66 | 17.39 | 98.47 | 88.76 | 68.12 | 22.06 | 98.36 | 91.35 |
| | Blast | 75.98 | 49.69 | 88.96 | 77.76 | 73.09 | 48.62 | 89.14 | 80.50 |
| Human | Boosted Trees | 78.65 | 28.16 | 92.32 | 78.01 | 78.70 | 27.06 | 93.95 | 80.61 |
| | Blast | 81.74 | 62.53 | 86.94 | 76.11 | 82.70 | 69.89 | 86.48 | 81.82 |
| Mouse | Boosted Trees | 80.24 | 25.00 | 92.50 | 76.56 | 80.08 | 22.41 | 93.83 | 78.13 |
| | Blast | 79.71 | 58.82 | 84.34 | 74.13 | 80.91 | 64.66 | 84.79 | 80.34 |

# Robert Ezra Langlois

## Education

*Ph.D., Bioinformatics*, December 10, 2007
University of Illinois at Chicago
Chicago, IL

*B.S., Bioengineering*, May 2002
University of Illinois at Chicago
Chicago, IL

## Publications

### Journal Papers

1. Bhardwaj, N., **Langlois, R. E.**, Zhao, G., and Lu, H.: Kernel-based machine learning protocol for predicting DNA-binding proteins. Nucleic Acids Research, 33(20):6486–6493, 2005.
2. Bhardwaj, N., Stahelin, R. V., **Langlois, R. E.**, Cho, W., and Lu, H.: Structural bioinformatics prediction of membrane-binding proteins. Journal of Molecular Biology, 359(2):486–495, 2006.
3. **Langlois, R.**, Carson, M., Bhardwaj, N., and Lu, H.: Learning to translate sequence and structure to function: Identifying DNA binding and membrane binding proteins. Annals of Biomedical Engineering, 35(6):1043–1052, 2007.
4. **Langlois, R. E.**, Diec, A., Perisic, O., Dai, Y., and Lu, H.: Improved protein fold assignment using support vector machines. International Journal of Bioinformatics Research and Applications, 1(3):319–334, 2006.
5. Ye, Z.-Q., Zhao, S.-Q., Gao, G., Liu, X.-Q., **Langlois, R. E.**, Lu, H., and Wei, L.: Finding new structural and sequence attributes to predict possible disease association of Single Amino acid Polymorphism (SAP). Bioinformatics, 23(12):1444–1450, 2007.

### Conference Papers

1. Bhardwaj, N., **Langlois, R. E.**, Zhao, G., and Lu, H.: Structure based prediction of binding residues on DNA-binding proteins. In 26th Annual International Conference on the IEEE EMBS, pages 2611–2614, Shanghai, China, 2005.
2. **Langlois, R. E.**, Diec, A., Dai, Y., and Lu, H.: Kernel based approach for protein fold prediction from sequence. In 26th Annual International

Conference on the IEEE EMBS, volume 2 of Engineering in Medicine and Biology Society, pages 2885–2888, San Francisco, CA, 2004. IEEE.

**Conference Presentations**

1. **Langlois, R.** and Lu, H.: Elucidating the characteristics of DNA binding proteins. Biophysical Journal, Supplement, Abstract, 1621-Plat, 2006. Platform for the 50th Annual Meeting of the Biophysical Society in Salt Lake City, Utah, February 18-22, 2006.
2. **Langlois, R. E.** and Lu, H.: Predicting the structure of non-homologous proteins. In Sigma Xi at University of Illinois at Chicago, Chicago, IL, 2005. Presentation for Sigmaxi 2005.

**Conference Posters**

1. Carson, M. B., **Langlois, R.**, Bhardwaj, N., and Lu, H.: Performance assessment of various machine learning classifiers in the prediction of DNA-binding and membrane-binding proteins. Biophysical Journal, Supplement, Abstract, 255-Pos, 2007. Poster for the 51st Annual Meeting of the Biophysical Society in Baltimore, Maryland, March 3-7, 2007.
2. Feng, G., **Langlois, R.**, and Lu, H.: Computer modeling of force-induced protein unfolding and unbinding. Biophysical Journal, Supplement, Abstract, 712-Pos, 2006. Poster for the 50th Annual Meeting of the Biophysical Society in Salt Lake City, Utah, February 18-22, 2006.
3. **Langlois, R.** and Lu, H.: A kernel based recognition method for classifying multiple folds. Biophysical Journal, Supplement, Abstract, 1580-Pos, 2004. Poster for the 48th Annual Meeting of the Biophysical Society in Baltimore, Maryland, February 14-18, 2004.
4. **Langlois, R.** and Lu, H.: Multi-resolution simulation of reversible unfolding of mechanical proteins. Biophysical Journal, Supplement, Abstract, 2742-Pos, 2005. Poster for the 49th Annual Meeting of the Biophysical Society in Long Beach, California 2005.
5. **Langlois, R.** and Lu, H.: Structure modeling of PKCs' and their binding subtrates. Biophysical Journal, Supplement, Abstract, 2444-Pos, 2005. Poster for the 49th Annual Meeting of the Biophysical Society in Long Beach, California 2005.
6. **Langlois, R.** and Lu, H.: Deciphering function from sequence and structure: Identifying non-homologous DNA-binding proteins using machine learning. Biophysical Journal, Supplement, Abstract, 2522-Pos, 2007. Poster

for the 51st Annual Meeting of the Biophysical Society in Baltimore, Maryland, March 3-7, 2007.

7. Zhao, G., Bhardwaj, N., **Langlois, R.**, and Lu, H.: Structural bioinformatics prediction of transcription factors and target sites. Biophysical Journal, Supplement, Abstract, 1997-Pos, 2005. Poster for the 49th Annual Meeting of the Biophysical Society in Long Beach, California 2005.

**Papers In Preparation**

1. **Langlois, R.** and Lu, H.: Boosting the prediction and understanding of DNA-binding proteins, 2007.
2. **Langlois, R.** and Lu, H.: Classification to multiple-instance learning using AdaBoost, 2007.
3. **Langlois, R.** and Lu, H.: Comparative proteomics with machine learning, 2007.
4. **Langlois, R.** and Lu, H.: malibu: A machine learning workbench, 2007. `http://proteomics.bioengr.uic.edu/malibu/index.html`.

## Memberships

Biophysical Society

## Awards

Travel Award ICAM Workshop 2006

Travel Award Machine Learning Summer School 2005

## Employment

8/2006–1/2007 Teaching Assistant
Laboratory of Computational Proteomics, University of Illinois at Chicago
Chicago, IL

8/2004–8/2006 Research Fellowship
Training in Cellular Signalling (P.I. John Solaro)
Laboratory of Computational Proteomics, University of Illinois at Chicago
Chicago, IL

5/2003–8/2004 Research Assistant
Laboratory of Computational Proteomics, University of Illinois at

Chicago
Chicago, IL

5/2001–8/2001 Statistical Informatics Intern
Motorola
Northbrook, Illinois

1/1999–5/1999 Web Developer
Criminal Justice Department, University of Illinois at Chicago
Chicago, IL

12/1999–1/2000 Assistant Event Coordinator
America's Millennium
Wachington D.C

5/1999–8/1999 Marketing/Engineering Intern
Motorola
Beijing, China

5/1998–8/1998 Engineering Intern
Motorola
Rolling Meadows, Illinois

12/1997–1/1998 Engineering Intern
Motorola
Rolling Meadows, Illinois

5/1997–8/1997 Engineering Intern
Motorola
Schaumburg, Illinois

## Conferences

10/2007 Midwest Symposium on Computational Biology and Bioinformatics
Northwestern University, Chicago, Illinois

3/2007 Midwest Grid Workshop
University of Illinois at Chicago, Chicago, Illinois

3/2007 51st Annual Meeting of the Biophysical Society
Baltimore, Maryland
2 Posters

12/2006 RECOMB Systems Biology
University of California, San Diego, California

2/2006 50th Annual Meeting of the Biophysical Society

| | |
|---|---|
| | Salt Lake City, Utah<br>Platform and Poster |
| 6/2006 | International Conference on Machine Learning<br>Carnegie Mellon University in Pittsburgh, Pennsylvania |
| 5/2006 | ICAM Workshop – **Travel Award**<br>Washington University, St. Louis, Missouri |
| 6/2005 | Modeling Protein Interactions<br>University of Kansas, Lawrence, Kansas |
| 5/2005 | Machine Learning Summer School – **Travel Award**<br>University of Chicago, Chicago, Illinois |
| 5/2005 | Midwest Protein Fold Conference<br>University of Notre Dame, South Bend, Indiana |
| 2/2005 | 49th Annual Meeting of the Biophysical Society<br>Long Beach, California<br>3 Posters |
| 2/2004 | 48th Annual Meeting of the Biophysical Society<br>Baltimore, Maryland<br>1 Poster |