Using machine learning to predict gene expression and discover sequence motifs

Xuejing Li

Submitted in partial fulfillment of the

requirements for the degree

of Doctor of Philosophy

of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Using machine learning to predict gene expression and discover sequence motifs

Xuejing Li

Recently, large amounts of experimental data for complex biological systems have become available. We use tools and algorithms from machine learning to build data-driven predictive models. We first present a novel algorithm to discover gene sequence motifs associated with temporal expression patterns of genes. Our algorithm, which is based on partial least squares (PLS) regression, is able to directly model the flow of information, from gene sequence to gene expression, to learn cis regulatory motifs and characterize associated gene expression patterns. Our algorithm outperforms traditional computational methods e.g. clustering in motif discovery.

We then present a study of extending a machine learning model for transcriptional regulation predictive of genetic regulatory response to *Caenorhabditis elegans*. We show meaningful results both in terms of prediction accuracy on the test experiments and biological information extracted from the regulatory program. The model discovers DNA binding sites *ab intio*. We also present a case study where we detect a signal of lineage-specific regulation.

Finally we present a comparative study on learning predictive models for motif discovery, based on different boosting algorithms: Adaptive Boosting (AdaBoost), Linear Programming Boosting (LPBoost) and Totally Corrective Boosting (TotalBoost). We evaluate and compare the performance of the three boosting algorithms via both statistical and biological validation, for hypoxia response in *Saccharomyces cerevisiae*.

Contents

List of Tables

List of Figures

Acknowledgments

First, I would like to express my deep and sincere gratitude to my thesis advisor Chris Wiggins. I have benefited incredibly from his invaluable support and encouragement in my research projects. I am also deeply grateful to Professor Christina Leslie for her broad and deep views in the field of machine learning and her generosity in sharing research ideas. I would also like to thank my other co-workers Anshul Kundaje, Steve Lianoglou, Valerie Reinke, Casandra Panea and Marta Arias.

I would like to thank all professors in the physics department teaching physics classes during my first two years at Columbia. They are Aleiner Igor, Blaer Allan, Gyulassy Miklos, Hui Lam, Mawhinney Robert, Mueller Alfred, Pinczuk Aron, Millis Andrew. I also appreciate the invaluable help from our physics department staff Lalla Grimes, Nicole Griggs and Joey Cambareri. I would like to thank my graduate friends at Columbia including Hui Zhou, Yue Zhao, Jie Lin, Min Li, Jiaming Jing, Xiao Wei, Daohua Song, Yang Wu, Shu Li, Sheng Wang, Qongying Hu and Dora Tan. They all have contributed to my meaningful life at Columbia.

Last and most importantly, I want to thank my family. I am deeply indebted to my parents for their love and care. Without their endless support I could not have gone this far. I owe my special thanks to my husband Si Li. He has always been encouraging me during my academic education and pursuit of career.

Chapter 1

Introduction

Scientific modeling centers on testing theory against experimental observations. Therefore experimental data plays an important role in constructing and testing theories in fields including astrophysics, high energy physics, biology and medicine. With the advent of computers and information age, huge amounts of data are being generated that change the way scientists perform research. The easy access to experimental data makes the data issue no longer the limit factor for research. Instead, the challenge now is how to extract the right information and understand "what the data says". This is called *learning from data*.

Given vast amounts of experimental data, research scientists could try to study and understand how complex systems behave as a whole rather than to focus on isolated components. Scientists can now simultaneously monitor interaction between individual components from experimental data. For example, microarrays in biology can measure the expression levels of thousands of genes at once in the tissue samples. And the gene expression data can be used to model the transcriptional regulatory network, which is characterized by interactions between thousands of genes and proteins.

Machine learning is a young scientific discipline concerned with the generation of computer algorithms based on statistical models learned from data. Those computer algorithms perform tasks of pattern recognition, classification, and prediction based on information from data.

Machine learning is data driven. However, it is more than traditional data analysis. The focus is on generating hypotheses and making accurate predictions for new data instead of simply describing existing data. In machine learning, every model has a certain prediction accuracy. The higher the prediction accuracy, the better the model is.

Like traditional scientific models, machine learning algorithms also aim to achieve the following two goals: first, it accurately describes existing experimental observations. second, it can make predictions for future observations. However, machine learning focuses more on the predictive modeling part. For example, many of the computational algorithms learn mathematical models by optimizing performance on unseen data.

Machine learning algorithms usually generate empirical hypotheses rather than theories of physical laws. In traditional scientific modeling approaches, scientists compare experimental data with a theory based on a fundamental and mathematical description of physical laws. The comparison validates the theory in the form of a mathematical formula given experimental data. Or the data can be used to fit a mathematical model by determining physical parameters. In contrast, machine learning algorithms do not lead to general fundamental theories. However it is very valuable if there are large amounts of data with noisy patterns. Its goal is to obtain a theoretical generalization from the data by building empirical models and learning from examples. The results may not give a fundamental description of the natural phenomenon, but it is able to give a good understanding of the phenomenon under study by accurately predicting for future experiments. For example, in astrophysics astronomical surveys of the night sky are taken using thousands of photographic plates, which contain about millions of objects. After the attributes of each object are measured, the scientific problem is to classify each object as a type of star galaxy. Since there are both a huge number of objects and features of objects, traditional scientific models are inapplicable. The decision-tree learning algorithms in machine learning have been successfully used to learn predictive models classifying objects.

This thesis presents three case studies of machine learning applications in biophysical modeling. First in Chapter 2 we give an introduction to machine learning, computational algorithms used for the work in this thesis and basic biological terminology.

In Chapter 3 we then present a novel algorithm to discover gene sequence motifs associated with temporal expression patterns of genes. Traditional computational methods have focused on clustering genes by expression first and then applying motif discovery algorithms to gene sequences of each cluster. However the clustering approach is based on the assumption that correlation of expression implies coregulation, which oversimplifies the biology of transcriptional regulation. Our algorithm, which is based on partial least squares (PLS) regression, is able to directly model the flow of information, from gene sequence to gene expression, to learn cis regulatory motifs and characterize associated gene expression patterns. To evaluate the performance of our algorithm, we use the mRNA profiling experiments from developmental time courses for worm *Caenorhabditis elegans*, a key model organism in developmental biology. This work has been published in Li et al. (2010).

In Chapter 4 we present the MEDUSA algorithm for learning regulatory programs that accurately predict gene expression, which was published in Kundaje et al. (2007). In Chapter 4 we extend the machine learning algorithm to *Caenorhabditis elegans* and we show meaningful results both in terms of prediction accuracy on the test experiments and biological information extracted from the regulatory program. The model discovers DNA binding sites *ab intio*. And despite the fact that the expression data comes from the whole embryo samples, we show that it reveals context-specific regulation by presenting a case study where we detect a signal of lineage-specific regulation.

In Chapter 5 we present how to learn predictive models for motif discovery based on different boosting algorithms: Adaptive Boosting (AdaBoost), Linear Programming Boosting (LPBoost) and Totally Corrective Boosting (TotalBoost). We evaluate and compare the performance of the three boosting algorithms via both statistical and biological validation, for hypoxia response in *Saccharomyces cerevisiae*.

Chapter 2

Background

2.1 Learning from data

2.1.1 Introduction to Machine Learning

Machine learning is a scientific discipline that studies algorithms capable of automatically extracting information from empirical data. Machine learning answers the question of how to build learning algorithms that automatically improve with experience or information, and performs tasks of gaining knowledge, making predictions, making decisions or building models based on given input data. A learner in a machine learning algorithm takes advantage of the data to reveal characteristics of certain unknown underlying probability distributions. Data could be seen as examples that contain information for relations between variables of interest. For example, a machine learning algorithm could focus on automatically recognizing complex patterns from observed examples also known as training data and making predictions on new examples. The challenge is that the complete set of all possible behaviors could be too large to be covered by the information in the training data. Hence the difficulty for a learning task is to have good generalization from the observed examples, in order to be able to produce a useful model for new examples.

Machine learning is the scientific field at the intersection of Computer Science and Statistics. Computer Science seeks to answer the question like “How can we build programs that solve problems in an automated way”. While Statistics looks at the problem of statistical inference from data and modeling of data given certain assumptions. Machine learning is closely related to statistical inference, data mining and pattern recognition. Many of the machine learning algorithms have been successfully applied to a broad range of scientific and engineering problems, such as the tasks of building customized search engines, recognizing handwritten digits, designing mobile robots, predicting stock price moves and classifying cancel tissues. In a typical scenario, the outcome measurements could be quantitative (e.g. stock price) or categorical (e.g. cancer or not cancer), which we want to predict based on a set of predictors (e.g. clinical measurements) for examples (e.g. people).

One application of machine learning algorithms in physics is for particle event selection in High Energy Physics (HEP). Experimental high energy physicists use modern accelerators, which collide protons and/or anti-protons to create new particles. The majority of accelerator events do not produce particles of interest, e.g. quark or Higgs boson. Therefore good data analysis is needed for effective event selection to separate events producing particles of interest (signal) from those producing other particles (background). Several supervised learning methods have been used to train classifiers that separate signal from background. Neural network was first brought to applications in HEP. For example, D0 collaboration uses neural networks to search for single top quark production Hopfield (1982); Abazov et al. (2001). More recently, new machine learning algorithms such as support vector machines and boosting are introduced for HEP analysis. For example, Whiteson and Naumann (2003) presents examples of applying support vector regression to the search for a computationally predicted particle and to the classification of heavy quark jets. It also compared the performance of the support vector machine with that of neural network. A variant of Boosting called Boosted Decision Tree was first introduced for data analysis in the MiniBooNE experiment at Fermi National Accelerator Laboratory, which was designed to detect signal for $\nu_\mu \rightarrow \nu_e$ oscillations Roe et al. (2005). The study showed that particle identification with the boosting algorithm is 20-80% better than standard neural network technique.

Learning problems come in two categories: supervised and unsupervised. In supervised learning (also known as directed data mining), the variables of interest are split into two groups: explanatory variables and dependent variables. The goal of the supervised learning is to identify a relationship between the explanatory variables (inputs) and the response variables (outputs), and generates a function that maps inputs to outputs. For example, there are algorithms for supervised learning of classification and regression functions. In these cases we have the labeled training examples in pairs $\{(\mathbf{x}_i, \mathbf{y}_i), i = 1, ..., n\}$ of inputs $\mathbf{x}_i$ and outputs $\mathbf{y}_i$. Outputs $\mathbf{y}_i$ are quantitative for regression problems and categorical for classification problems. A predictive function f which maps from $\mathbf{x}_i$ to $\mathbf{y}_i$ is to be learned and we can compare $f(\mathbf{x}_i)$ with $\mathbf{y}_i$ to estimate the error rate of the prediction function. For example, $\mathbf{x}_i$ may be a image vector and $\mathbf{y}_i$ the qualitative label of the object in the image for training an image recognition program. Supervised learning algorithms such as support vector machines for classification may be used to estimate the predictive function f from the data.

In unsupervised learning problems there are response variables to be predicted. All training examples only have $\mathbf{x}_i$ and the target of the analysis could vary from reducing the data dimension to finding patterns in the distribution of $\mathbf{x}_i$. Since the examples are unlabeled, there is no direct error function to evaluate the learning performance. An example of unsupervised learning is clustering, e.g., cluster genes based on their time series expression. Clustering analysis groups a set of examples into subsets (called clusters) so that examples in the same cluster are similar based on a predefined distance measure. Principal component analysis (PCA) is another example of unsupervised learning. PCA find a small set of factors that explain the significant variance of $\mathbf{x}_i$ and can also be combined to reconstruct the data.

In summary, supervised learning algorithms have the response variables to guide the learning procedure and evaluate the learning performance. While in the unsupervised learning problems, without variables to predict task of the analysis is to find the hidden structure of the data or cluster the data given a predefined measure. In our work we will apply and extend two supervised learning algorithms: partial least squares regression for regression problems and boosting for classification problems.

2.1.2 Model Selection

The problem of model selection is an essential part of every learning problem. Given some data, machine learning algorithms could learn a number of models that fit the data well and one may have no preference for one model over another before certain assumptions are made. Model selection involves choosing a model that best selects features and parameters to create a model of optimal complexity for data. One basic rule is that if models fit the data equally well, a simpler model is preferred Zellner and Keuzenkamp (2001). For example, *Occam's razor* suggests that when theories have equal predictive power, one should choose the theory with the fewest assumptions Hoffmann et al. (1997) .

We now illustrate the problem of model selection, which is important to every learning task, using the example of classical polynomial curve fitting. We assume our data points are sampled from a distribution of polynomial order of three with Gaussian noise as shown in Figure 2.1. There are ten data points. Because it is a Gaussian distribution, maximizing the likelihood of those data points is equivalent to minimizing mean squared error. We try to fit polynomials of different degrees (M). Curves fitted for polynomial order $M = 0, 1, 3, 9$ are shown in the Figure 2.1. Mean squared error decreases with higher polynomial degree M and polynomial order of $M = 9$ fits the training dat best. Generally as we use more and more complex and flexible models, the mean squared error will get smaller and model better fits the training data. However, the problem is to choose the best order M for the polynomial function that not only fits the observed data, but also generalizes well to new data. For a polynomial fitting of order $M = 9$, we see significant fluctuations in the values fitted. The reason is that the higher-order polynomial model, with more parameters, is fitting the noise in the data which will not be repeated in new data sampled from the distribution. In other words, the more complicated models overfit the data. We also note that polynomial order of $M = 0$ and $M = 1$ also perform worse the $M = 3$. The problem is not overfitting but underfitting. In other words, the model is not complicated enough to capture the patterns in the data. Therefore the problem of model selection accompanies every learning task: find the model of optimal complexity if we have a choice of models.

Figure 2.2 illustrates an important aspect of model performance evaluation and selection. We first consider a continuous response variable Y and inputs $\mathbf{X}$ for a regression problem. If we assume that $\mathbf{Y} = f(\mathbf{X}) + \epsilon$, where $E(\epsilon) = 0$ and $Var(\epsilon) = \sigma_\epsilon^2$. We obtain a predictive model $\hat{f}(\mathbf{X})$ learning from training examples. A typical loss function is squared error $(\mathbf{Y} - \hat{f}(\mathbf{X}))^2$ and training error is the average loss over training examples

$$Err_{training} = \frac{1}{N}\sum_{i=1}^{N}((\mathbf{y}_i - \hat{f}(\mathbf{x}_i))^2)$$

The best measure for the generalization capability of the estimated model $\hat{f}$ is error of prediction, also known as test error. Prediction error is the expected loss over new test examples.

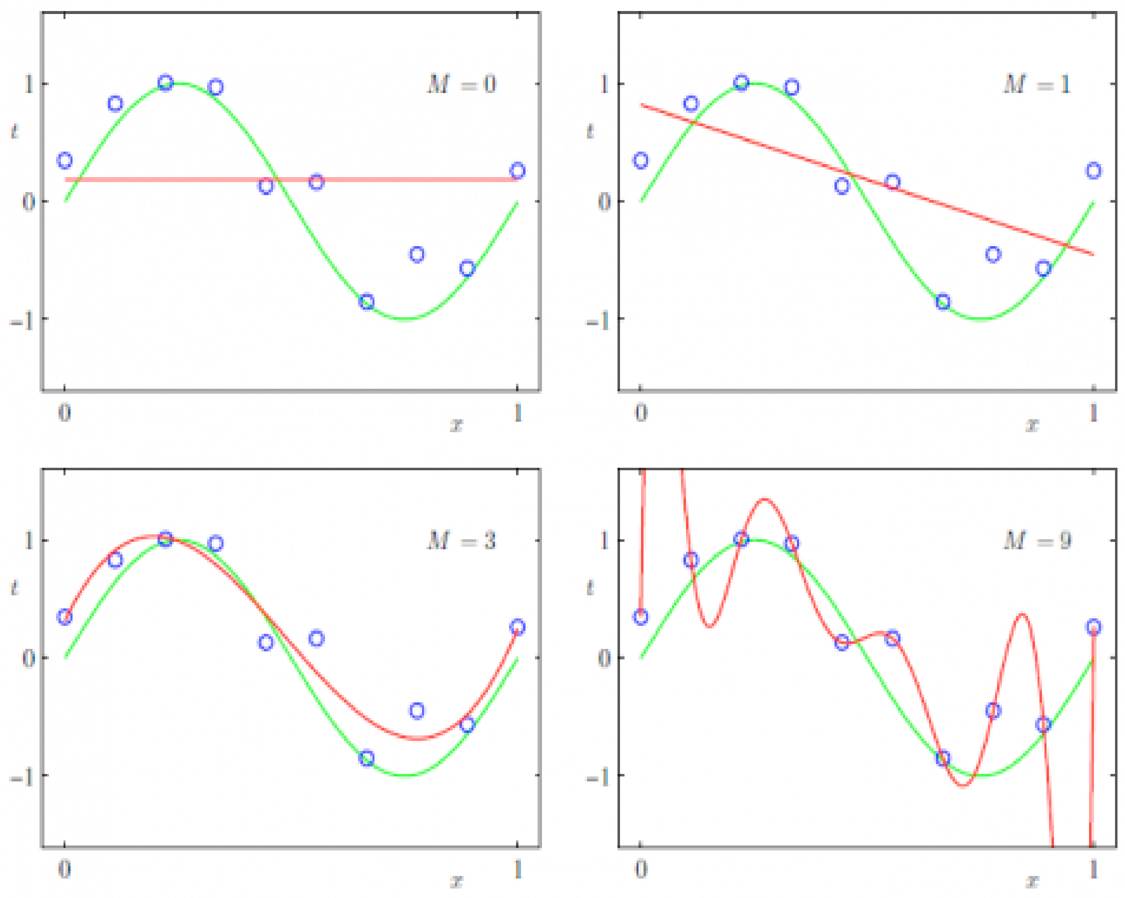


Figure 2.1: **Fitting a data set to different orders of polynomials (Bishop,"Pattern Recognition and Machine learning").**

$$
\begin{aligned}
Err_{test} &= E_{test}((\mathbf{Y} - \hat{f}(\mathbf{X})^2) \\
&= \sigma_\epsilon^2 + [E\hat{f}(\mathbf{X}) - f(\mathbf{X})]^2 + E[\hat{f}(\mathbf{X}) - E\hat{f}(\mathbf{X})]^2 \\
&= \sigma_\epsilon^2 + Bias^2(\hat{f}(\mathbf{X})) + Var(\hat{f}(\mathbf{X})) \qquad (2.1)
\end{aligned}
$$

The first term is variance around its true mean $f(\mathbf{X})$, which can not be avoided. The second term measures the difference between the average of our estimate and the true mean, which is often referred to as the bias term. The last one is variance, average deviation of $\hat{f}(\mathbf{X})$ around its own mean. Therefore, the error of prediction that we can control is composed of two main parts: bias and variance Martens and Naes (1992). Bias is high as the estimated model is not complex enough to capture all the relationships between responses and predictors. High variance is caused by modeling random noise in the data. As the model becomes more complex, it is able to capture more complicated structures in the data leading to a decrease in bias. However the variance could increase. Increasing prediction error due to a too complex model is called overfitting or overtraining while increasing prediction error due to a too simple model is called underfitting. Therefore bias and variance need to balance each other to obtain an low prediction error (arrow in figure 2.2). In other words, there exists a model with optimal complexity that gives minimum test error.

There are a number of methods for estimating how accurately a predictive model will perform. Typically the model has one or a set of parameters, which vary with the complexity of the model. We hope to find the parameter values that minimize test error giving the optimal model complexity in figure 2.2. One simple and popular method for estimating minimum test error is cross-validation. Cross-validation is particularly useful for small data sets, because if we have enough data we could easily have a validation set to assess the prediction performance. If there is insufficient data to split data into one training set and one validation set, K-fold cross-validation repeats the process of training and validation K times and combines the K estimates of test error. More specifically, all data is randomly divided into K equally-sized subsamples and of the K subsamples, one subsample is set aside as the validation data and the remaining K-1 subsamples are used as training data. This cross-validation process is then repeated K times (the folds) and each of the K subsamples used exactly once as the validation set. The K results from the K folds are then averaged to produce a single estimation of test error. 10-fold or 5-fold cross-validation are commonly used. The case in which K equals the

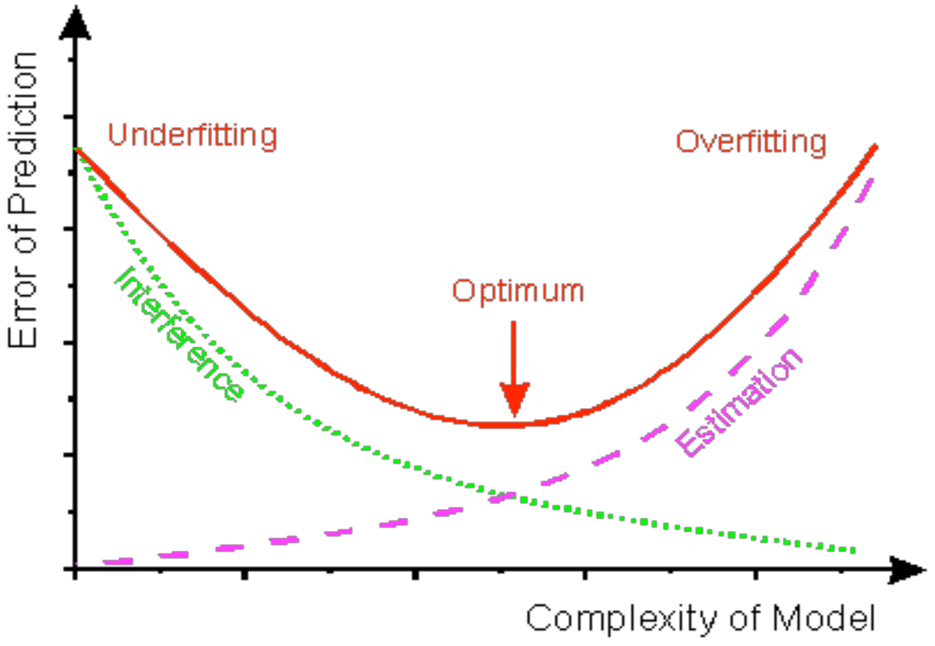


Figure 2.2: **Behavior of prediction error as a function of model complexity. (Dr. Frank Dieterle, Ph.D. thesis)**

number of examples is known as *leave-one-out* cross-validation. In *leave-one-out* cross-validation, for the ith observation the model is fit to all the data except the ith subsample.

We often assume a type of underlying distribution of the data for regression problems, e.g. Gaussian noise. For classification problems in which the prediction is in $\{-1, +1\}$, a "distribution-free" framework for learn has been introduced in the field of computational learning theory Valiant (1984). The labels -1 and $+1$ define two classes for the classifier to predict. The "distribution-free" framework considers a domain X and a set C of functions (or concepts) c: $X \rightarrow \{-1, +1\}$. The data is sampled from an unknown probability distribution $p(x$. A machine learning algorithm outputs a hypothesis h: $X \rightarrow \{-1, +1\}$ from a set H of candidate hypotheses.

We call the concept set learnable, if there exists an algorithm B and a polynomial function $m(.,.)$ such that for any c in C, any distribution $p(x)$, any $\epsilon > 0$ and $\delta > 0$, algorithm B takes more than $m(\frac{1}{\epsilon}, \frac{1}{\delta})$ training examples to produce a hypothesis for which the probability of generalization error greater than ϵ is smaller than δ. The framework is also called *PAC* learnable for Probably Approximately Correct (PAC) learning Kearns and Valiant (1988). In the PAC framework, we obtain a bound on the generalization error. In Vapnik's theory work Vapnik (1995), it is shown that with probability $1-\eta$ the generalization error has an upper bound:

$$\text{generalization error} \leq \text{training error} + \sqrt{\frac{d(log(2N/d)+1) - log(\eta/4)}{N}} \quad (2.2)$$

N is number of data points and d is called the *VC dimension* Vapnik (1995), which is a measure of complexity of the hypothesis space. Training error will decrease as the model complexity increases. However the term related to VC dimension will increase, which offsets the smaller training error for the upper bound on generalization error. This upper bound is also used for the purpose of model selection. In other words, choose a model that minimizes the bound. This method for model selection is known as *structural risk minimization.*

2.1.3 Partial Least Squares Regression

2.1.3.1 Introduction to Partial Least Square Regression

Scientific research often uses controllable continuous variables (factors or predictors) to explain and predict the behavior of other continuous variables (responses). If the relationship between predictor variables and response variables is known to be linear, a linear regression model is constructed to predict responses:

$$\begin{aligned}
\mathbf{y}_1 &= b_{11}\mathbf{x}_1 + b_{12}\mathbf{x}_2 + ... + b_{1p}\mathbf{x}_p \\
... &= \quad ... \\
\mathbf{y}_q &= b_{q1}\mathbf{x}_1 + b_{q2}\mathbf{x}_2 + ... + b_{qp}\mathbf{x}_p
\end{aligned}$$

where $\mathbf{x}_1, ..., \mathbf{x}_p$ are predictor variables and $\mathbf{y}_1, ..., \mathbf{y}_q$ are response variables. The derived $\mathbf{b}_{ij}(i = 1, ..., q, j = 1, ..., p)$ are regression coefficients. Note that the emphasis is on accurate prediction of responses and thus linear regression techniques used for this purpose are in the category of machine learning. We refer to it as 'learning' because we would like to 'learn from data' to discover the linear relationship between the set of response variables we want to predict and other independent predictor variables. We use our 'data' to train a machine learning algorithm and then use the learned statistical model to predict future observations.

When the predictors are only a few and have a known relationship with the responses, ordinary multiple regression can be easily used to build a predictive model. However, if the number of predictors gets too large, ordinary multiple regression is can be inefficient or infeasible. For example, it is likely to yield a model that fits the samples perfectly but is unable to predict new data well. This is known as over-fitting. Although there are many factors learned, they may be poorly related to response variables and thus do not account for the variation in the responses. poreover, when there are more predictors than observations, ordinary multiple regression may not work if predictor variables are highly correlated. This is known as multicollinearity and can make it difficult to perform the matrix inversion required in ordinary multiple regression.

To address those data analysis problems, the ordinary multiple regression model has been extended in a few ways. The ordinary multiple regression model serves as a foundation for a number of other multivariate methods. For example, principal components regression predicts responses on the dependent variables from factors underlying the levels of predictor variables. Canonical correlation predicts factors underlying responses on the dependent variables from factors underlying the levels of the predictor variables. One common feature of those multivariate regression methods is that they extract factors underlying $\mathbf{Y}$ and $\mathbf{X}$ variables from $\mathbf{Y}^T\mathbf{Y}$ and $\mathbf{X}^T\mathbf{X}$ respectively. Those factors are selected to explain the variation in $\mathbf{Y}$ and $\mathbf{X}$ separately instead of cross-product matrices containing both the $\mathbf{Y}$ and $\mathbf{X}$ variables. Therefore factors underlying predictor variables are not necessarily relevant for response variables. In this section, we present a popular regression technique known as partial least squares (PLS) regression that addresses this issue. The general idea of PLS regression is to try to extract latent factors that account for predictor variation and model the responses well as well. Specifically, PLS regression is a method of dimension reduction combined with ordinary multiple regression. In PLS regression, the prediction of response variables can be first achieved by learning a set of orthogonal factors called latent factors from given predictors. The latent factors are chosen such that the covariance between response variables and predictor variables is maximal. In short, factors in PLS regression are selected to reflect the covariance structure between the predictor and response variables, while other regression methods like principal components regression produces factors reflecting the covariance structure between the predictor variables. Therefore different from those regression methods, PLS regression extracts latent factors with the best predict power for response variables.

The PLS regression was first proposed by Herman Wold in the 1960s and 1970s to solve problems in the econometric path modeling in the social sciences Wold (1966; 1975) and was later applied to regression problems in chemometrics by his son Svante Wold Wold et al. (1984). PLS started to get popular in the statistics field about 15 years ago because the method was found to work very well for data with small sample sizes and a very large set of predictors. Since then there have been successful applications of PLS regression in scientific areas including bioinformatics, physiology and medicine Nguyen and Rocke (2002; 2004).

PLS regression is a powerful analysis tool for high-dimensional data with many continuous response variables. PLS regression works particularly well for a small number of samples with a large set of non-independent predictors Jong (1993); ter Braak and de Jong (1998). It is computationally efficient and it reveals meaningful structures in both predictors and responses via selection of latent factors. However, there exists a large number of variants of PLS algorithms, which makes it very difficult to understand the principles behind PLS regression. Here we give a overview of known PLS methods and their underlying mathematics.

The remainder of this section is organized as follows. In 'Mathematical Foundations of PLS Regression' subsection, we explain the general methodology of PLS regression. In 'Univariate Response: PLS1' subsection, we review the mathematics of the PLS variant PLS1 for the case of univariate response variables. 'Multivariate Response: SIMPLS' subsection is devoted to the PLS variant SIMPLS for multivariate responses.

2.1.3.2 Mathematical Foundations of PLS Regression

Partial least squares (PLS) is a method for constructing predictive models when there are many highly correlated predictors. In this section, we give a brief overview of how PLS regression works and compare it with other regression techniques such as principal components regression. Suppose we have independent variables (predictors) in the matrix form as $\mathbf{X}$ (dimension $n \times p$), where n is number of observations and p is number of predictors. And we have response variables in the matrix form as $\mathbf{Y}$ (dimension $n \times q$), where q is number of response variables. We call $\mathbf{X}$ the input matrix and $\mathbf{Y}$ the output matrix. qach column in $\mathbf{X}$ is a predictor $x_i = (x_{1i}, ..., x_{ni})^T$ and $\mathbf{Y}$ is a response variable $y_i = (y_{1i}, ..., y_{ni})^T$:

$$\mathbf{X} = (x_1, ..., x_p)$$
$$\mathbf{Y} = (y_1, ..., y_q)$$

The goal of PLS regression is to identify a mathematical mapping from $\mathbf{X}$ to $\mathbf{Y}$ and then predict $\mathbf{Y}$ from $\mathbf{X}$. When there are more predictors than observations ($n < p$), ordinary multiple regression is not feasible because of multicollinearity in $\mathbf{X}$. Several regression methods have been proposed to address this problem. For example, a regression analysis known as principal components regression first reduces number of predictors by running a principal components analysis of the $\mathbf{X}$ matrix. The analysis selects principal components that best explain the variance in $\mathbf{X}$ matrix.

$$\mathbf{T} = \mathbf{X}\mathbf{W}$$

where columns of $\mathbf{T}$ are known as principal components and those of $\mathbf{W}$ as principal component loadings. The next step is to run ordinary multiple regression on the new principal components instead of the original predictors.

$$\mathbf{Y} = \mathbf{T}\mathbf{Q} + \mathbf{F}$$

Finally the regression coefficients are used to predict $\mathbf{Y}$ from principal components, which are simply linear combinations of columns in $\mathbf{X}$ matrix. Principal components regression does resolve the multilinearity problem in the data by reducing the original predictors to a set of orthogonal principal components. However, those principal components are chosen to explain $\mathbf{X}$ rather $\mathbf{Y}$. So there is no evidence that the selected principal components are relevant for $\mathbf{Y}$ and ideal variables to predict $\mathbf{Y}$. In contrast, PLS regression aims to extract factors in $\mathbf{X}$ that could explain as much as possible of the covariance between $\mathbf{X}$ and $\mathbf{Y}$ in the dimension reduction step. Therefore unlike principal components regression, the latent factors obtained by PLS regression are chosen given the input of response variables. To do this, PLS regression searches for a set of components called latent factors by performing a decomposition of $\mathbf{X}$ and $\mathbf{Y}$ simultaneously. By combining the information in both the predictors and response variables, PLS regression eliminates more irrelevant predictors for $\mathbf{Y}$. Similar to principal component regression, the dimension reduction step is followed by a regression step where latent factors of $\mathbf{X}$ are used to predict $\mathbf{Y}$.

PLS regression first scales $\mathbf{X}$ and $\mathbf{Y}$ so that each column of the input and output matrices x_i and y_i has zero mean and unit variance. When predictors outnumber observations $n < p$, the ordinary regression can not be applied because the covariance matrix of $\mathbf{X}$ is singular. In contrast, PLS regression may be feasible as it performs the following decomposition of both $\mathbf{X}$ and $\mathbf{Y}$ matrices.

$$\mathbf{Y} = \mathbf{T}\mathbf{Q}^T + \mathbf{F} \tag{2.3}$$

$$\mathbf{X} = \mathbf{T}\mathbf{P}^T + \mathbf{E} \tag{2.4}$$

where $\mathbf{T}$ is a $n \times K$ matrix giving K latent factors for the n observations, $\mathbf{P}$ (dimension $p \times K$) and $\mathbf{Q}$ (dimension $q \times K$) are coefficient matrices and $\mathbf{E}$ (dimension $n \times p$) and $\mathbf{F}$ (dimension $n \times q$) are matrices of random errors. PLS regression constructs the matrix of latent factors $\mathbf{T}$, which is a linear transformation of $\mathbf{X}$:

$$\mathbf{T} = \mathbf{X}\mathbf{W} \tag{2.5}$$

where $\mathbf{W}$ is a weight matrix (dimension $p \times K$). Each column $t_i = (t_{1i}, ..., t_{ni})^T$ in $\mathbf{T}$ is a linear combination of columns $x_i = (x_{1i}, ..., x_{ni})^T$ in $\mathbf{X}$ and the linear coefficients are provide by $\mathbf{W}$.

$$\begin{aligned} \mathbf{t}_1 &= w_{11}\mathbf{x}_1 + ... + w_{p1}\mathbf{x}_p, \\ ... &= \quad ... \\ \mathbf{t}_n &= w_{1n}\mathbf{x}_1 + ... + w_{pn}\mathbf{x}_p \end{aligned} \tag{2.6}$$

The latent factors in $\mathbf{T}$ then replace the original variables in $\mathbf{X}$ to predict $\mathbf{Y}$. Once $\mathbf{T}$ is found, we obtain $\mathbf{Q}$ by running ordinary least squares regression.

$$\mathbf{Q} = \mathbf{Y}^T\mathbf{T}(\mathbf{T}^T\mathbf{T})^{-1}$$

Finally, for PLS regression model $\mathbf{Y}$=$\mathbf{XB}$+$\mathbf{F}$ the matrix of regression coefficients $\mathbf{B}$ is derived

$$\mathbf{B} = \mathbf{W}\mathbf{Q}^T = \mathbf{W}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y}$$

and the fitted response matrix $\hat{\mathbf{Y}}$ is written as

$$\hat{\mathbf{Y}} = \mathbf{T}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y}$$

PLS regression outputs the matrix of regression coefficients $\mathbf{B}$ as well as matrices $\mathbf{W}$, $\mathbf{T}$, $\mathbf{P}$ and $\mathbf{Q}$. And $\mathbf{P}$ and $\mathbf{Q}$ are denoted as 'X loadings' and 'Y loadings', respectively. PLS regression takes into account the response $\mathbf{Y}$ when constructing $\mathbf{T}$ because decomposition of $\mathbf{X}$ and $\mathbf{Y}$ are performed simultaneously. $\mathbf{T}$ is learned such that it has high covariance with $\mathbf{Y}$. In this aspect PLS regression is a supervised learning method while principal components regression is not since it does not use the response variables for the construction of principal components. Therefore PLS regression usually performs better than principal components regression for prediction problems.

2.1.3.3 Univariate Response: PLS1

There are a number of variants of PLS, each of which defines and solves an optimization problem for constructing the weight matrix $\mathbf{W}$. Two important variants of PLS are PLS1 for one-dimensional response variables and SIMPLS for multi-dimensionl response. We first consider univariate response variables (q=1). $\mathbf{y}$ is a $n \times 1$ matrix or a vector of length n. Since data in matrices $\mathbf{X}$ and $\mathbf{y}$ are already centered, the covariance between the response variable $\mathbf{y}$ and latent factor $\mathbf{t}_i = w_{1i}\mathbf{x}_1 + ... + w_{pi}\mathbf{x}_p$ is computed as

$$Cov(\mathbf{y}, \mathbf{t}_i) = \frac{1}{n}\mathbf{w}_i^T\mathbf{X}^T\mathbf{y}$$

where $\mathbf{w}_i = (w_{1i}, ..., w_{pi})^T$ is a column in $\mathbf{W}$. Similarly, the variance of latent factor $\mathbf{t}_i$ is

$$VaR(\mathbf{t}_i) = \mathbf{w}_i^T\mathbf{X}^T\mathbf{X}\mathbf{w}_i$$

and the covariance between $\mathbf{t}_i$ and $\mathbf{t}_j$ $(i \neq j)$ is

$$Cov(\mathbf{t}_i, \mathbf{t}_j) = \frac{1}{n}\mathbf{w}_i^T\mathbf{X}^T\mathbf{X}\mathbf{w}_j$$

In PLS1, weight matrix $\mathbf{W} = (\mathbf{w}_1, ..., \mathbf{w}_K)$ is defined such that the covariances between $\mathbf{y}$ and latent factors $\mathbf{t}_i$ are maximal. For i=1,...,K, univariate PLS1 optimizes the following objective function:

$$\begin{aligned} \mathbf{w}_i &= argmax_{\mathbf{w}} Cov(\mathbf{y}, \mathbf{X}\mathbf{w})^2 \\ &= argmax_{\mathbf{w}} \mathbf{w}^T \mathbf{X}^T \mathbf{y}\mathbf{y}^T \mathbf{X}\mathbf{w} \\ s.t.& \ \ \mathbf{w}_i^T \mathbf{w}_i = 1 \\ & \ \ \mathbf{t}_i^T \mathbf{t}_j = \mathbf{w}_i^T \mathbf{X}^T \mathbf{X} \mathbf{w}_j = 0, \ for \ j = 1, ..., i-1 \end{aligned} \tag{2.7}$$

The latent factors $\mathbf{t}_i$ and weight vectors $\mathbf{w}_i$ are sequentially built by maximizing the covariance between the latent factors and the response subject to two constraints: first, the latent factors $\mathbf{X}\mathbf{w}_1, ..., \mathbf{X}\mathbf{w}_K$ are mutually uncorrelated; second, the weight vectors $\mathbf{w}_1, ..., \mathbf{w}_K$ have unit length. From a practical point of view, maximal covariance with the response indicates high predictive power and zero correlation means no redundancy in latent factors.

Using Lagrange's method, we first obtain $\mathbf{w}_1$

$$\mathbf{w}_1 = \mathbf{X}^T \mathbf{Y} / ||\mathbf{X}^T \mathbf{y}|| \tag{2.8}$$

PLS1 features 'deflation' of the matrix $\mathbf{X}$ for computing subsequent weight vectors $\mathbf{w}_2, ..., \mathbf{w}_K$

$$\mathbf{w}_i = \mathbf{X}_i^T \mathbf{y} / ||\mathbf{X}_i^T \mathbf{y}|| \tag{2.9}$$

where $\mathbf{X}_i = \mathbf{X}_{i-1} - \mathbf{t}_{i-1} \mathbf{t}_{i-1}^T \mathbf{X}_{i-1}$. A pseudocode description of PLS1 algorithm is given in Figure 2.3.

```
INPUT:
    $\mathbf{X}$ ($n \times p$, column normalized): matrix of predictors
    $\mathbf{y}$ ($n \times 1$, normalized): 1d response variables
    K: number of latent factors

Algorithm:
    $\mathbf{X}_1 \leftarrow \mathbf{X}$
    $\mathbf{w}_1 \leftarrow \mathbf{X}^T\mathbf{y}/||\mathbf{X}^T\mathbf{y}||$, an initial estimate of $\mathbf{w}$.
    $\mathbf{t}_1 \leftarrow \mathbf{X}\mathbf{X}^T\mathbf{y}/||\mathbf{X}^T\mathbf{y}||$, an initial estimate of $\mathbf{t}$.

    Loop over latent factors: for i = 1 to K
        $normt \leftarrow \mathbf{t}_i{}^T\mathbf{t}_i$
        $\mathbf{t}_i \leftarrow \mathbf{t}_i/normt$, normalize latent factor $\mathbf{t}_i$.
        $\mathbf{w}_i \leftarrow \mathbf{w}_i/normt$, adapt weights $\mathbf{w}_i$ accordingly.
        $\mathbf{p}_i \leftarrow \mathbf{X}_i{}^T\mathbf{t}_i$, get X loadings.
        $\mathbf{q}_i \leftarrow \mathbf{y}^T\mathbf{t}_i$, get y loadings.

        If $\mathbf{q}_i$= 0
            $K \leftarrow i$, break the for loop.
        If $l < K$, deflate $\mathbf{X}$
            $\mathbf{X}_{i+1} \leftarrow \mathbf{X}_i - l_i\mathbf{t}_i\mathbf{p}_i{}^T$
            $\mathbf{w}_{i+1} \leftarrow \mathbf{X}_{i+1}{}^T\mathbf{y}$
            $\mathbf{t}_{i+1} \leftarrow \mathbf{X}_{i+1}\mathbf{w}_{i+1}$
    End for

OUTPUT:
    Define $\mathbf{W}$ to be the matrix with columns $\mathbf{w}_1, ..., \mathbf{w}_K$. Similarly define $\mathbf{P}$,$\mathbf{Q}$.
    $\mathbf{B} \leftarrow \mathbf{W}\mathbf{Q}^T$ =, regression coefficients.
    Return B.
```

Figure 2.3: **Pseudocode for PLS1.**

2.1.3.4 Multivariate Response: SIMPLS

The multivariate response is more difficult as the latent factors have to explain all the responses $\mathbf{y}_1, ..., \mathbf{y}_q$ in matrix $\mathbf{Y}$ simultaneously. Since the original ideas behind the PLS decomposition were heuristic, a variety of PLS regression algorithms have been proposed Martens (2001). An important multivariate PLS regression algorithm is known as Statistically Inspired Modification of PLS (SIMPLS) Jong (1993). We suggest using the SIMPLS because it optimizes the same simple statistical criterion as PLS1. For i=1,...,K,

$$\begin{aligned} \mathbf{w}_i &= argmax_{\mathbf{w}} \mathbf{w}^T \mathbf{X}^T \mathbf{Y} \mathbf{Y}^T \mathbf{X} \mathbf{w} \\ &\quad s.t. \;\; \mathbf{w}_i^T \mathbf{w}_i = 1 \\ &\qquad\quad \mathbf{t}_i^T \mathbf{t}_j = \mathbf{w}_i^T \mathbf{X}^T \mathbf{X} \mathbf{w}_j = 0, \; for \; j = 1, ..., i-1 \end{aligned} \tag{2.10}$$

The objective function $\mathbf{w}^T \mathbf{X}^T \mathbf{Y} \mathbf{Y}^T \mathbf{X} \mathbf{w}$ is maximized by SIMPLS in the same way as in the univariate case by PLS1. The objective function could be rewritten as the sum of squared covariances between $\mathbf{T}$ and $\mathbf{y}_1, ..., \mathbf{y}_E$

$$\begin{aligned} \mathbf{w}^T \mathbf{X}^T \mathbf{Y} \mathbf{Y}^T \mathbf{X} \mathbf{w} &= ((\mathbf{X}\mathbf{w})^T \mathbf{Y})^T ((\mathbf{X}\mathbf{w})^T \mathbf{Y}) \\ &= n^2 \sum_{i=1}^{q} Cov(\mathbf{T}, \mathbf{y}_i)^2 \end{aligned} \tag{2.11}$$

where n is number of observation or number of rows in $\mathbf{X}$ and $\mathbf{Y}$. SIMPLS can be viewed as a generalization of univariate PLS to the case of multivariate response because it has the same optimality criterion $\mathbf{w}^T \mathbf{X}^T \mathbf{Y} \mathbf{Y}^T \mathbf{X} \mathbf{w}$ and the same constraints. In the actual SIMPLS procedure, the weights and latent factors are obtained by a GramSchmidt-type algorithm Jong (1993).

There is another equivalent objective function for SIMPLS that considers weight vectors for both the response variables and the predictor variables. Two sets of weights $\mathbf{w}$ in $\mathbf{X}$ space and $\mathbf{c}$ in $\mathbf{Y}$ space are found to create a linear combination of the columns in $\mathbf{X}$ and $\mathbf{Y}$ respectively such that their covariance is maximal. Specifically, it optimizes the following term for i=1,...,K

$$\begin{aligned}
(\mathbf{w}_i, \mathbf{c}_i) &= argmax_{\mathbf{w},\mathbf{c}} Cov(\mathbf{Xw}, \mathbf{Yc}) \\
&= argmax_{\mathbf{w},\mathbf{c}} \mathbf{w}^T \mathbf{X}^T \mathbf{Y}^T \mathbf{c} \\
s.t. &\quad \mathbf{w}_i^T \mathbf{w}_i = 1 \\
&\quad \mathbf{c}_i^T \mathbf{c}_i = 1 \\
&\quad \mathbf{t}_i^T \mathbf{t}_j = \mathbf{w}_i^T \mathbf{X}^T \mathbf{X} \mathbf{w}_j = 0, \ for \ j = 1, ..., i-1
\end{aligned} \tag{2.12}$$

The goal of the above SIMPLS formulation is to obtain a first pair of vectors $\mathbf{t}_1 = \mathbf{Xw}_1$ and $\mathbf{u}_1 = \mathbf{Yc}_1$ such that $\mathbf{t}_1^T \mathbf{u}_1$ is maximal subject to constraints that $\mathbf{w}_1^T \mathbf{w}_1 = 1$ and $\mathbf{c}_1^T \mathbf{c}_1 = 1$. After the first pair of latent vectors $\mathbf{t}_1$ and $\mathbf{u}_1$ are found, matrix $\mathbf{Y}^T\mathbf{X}$ is 'deflated' and this procedure is repeated to obtain all latent factors $\mathbf{t}$, $\mathbf{u}$ and weights $\mathbf{w}$, $\mathbf{c}$. The detailed SIMPLS pseudocode is given in Figure 2.4. This formulation suggests that PLS is related to classical canonical correlation analysis (CCA). The main difference between the two methods is that PLS maximizes covariances rather than correlations. Therefore PLS does not need to invert the covariance matrix, which is more appropriate for the analysis of high-dimensional data. It can be proven using results from linear algebra that the above formulation is equivalent to the original one CR (1993).

To summarize, SIMPLS has the following appealing features: first, it produces orthogonal, i.e. empirically uncorrelated, latent factors; second, it is easily generalized from PLS1 to multivariate response; finally, it optimizes a simple statistical criterion. Other PLS variants for multiple response variables in the literature have predictive power comparable to SIMPLS. However, some generate orthogonal weights $\mathbf{W}$ rather than orthogonal latent factors $\mathbf{T}$ Martens (2001). Or some do not easily generalize to multi-dimensional responses in terms of object function, e.g. Nonlinear Iterative Partial Least Squares (NIPALS) algorithm Geladi and Kowalski (1986). In addition, SIMPLS is also one of the most computationally efficient PLS algorithms.

```
INPUT:
    $\mathbf{X}$ ($n \times M$, column normalized): matrix of predictors
    $\mathbf{Y}$ ($n \times q$, column normalized): matrix of response variables
    $\mathbf{S} = \mathbf{Y}^T\mathbf{X}$: cross-product

Algorithm:
    $\mathbf{S}_1 = \mathbf{Y}^T\mathbf{X}$, initial value of cross-product
    Loop over latent factors:
    For i = 1 to K
        $\mathbf{q}_i =$ $dominant\ eigenvector\ of$ $\mathbf{S}_i\mathbf{S}_i^T$
        $\mathbf{w}_i = \mathbf{S}_i^T\mathbf{q}_i$, get X weights

        $\mathbf{t}_i = \mathbf{X}\mathbf{w}_i$, get X latent factors
        $normt \leftarrow \mathbf{t}_i^T\mathbf{t}_i$, compute norm
        $\mathbf{t}_i \leftarrow \mathbf{t}_i/normt$, normalize X latent factor $\mathbf{t}_i$.
        $\mathbf{w}_i \leftarrow \mathbf{w}_i/normt$, adapt weights $\mathbf{w}_i$ accordingly.

        $\mathbf{p}_i \leftarrow {\mathbf{X}_i}^T\mathbf{t}_i$, get X loadings.
        $\mathbf{c}_i \leftarrow \mathbf{Y}^T\mathbf{t}_i$, get Y weights.
        $\mathbf{u}_i \leftarrow \mathbf{Y}^T\mathbf{c}_i$, get Y latent factor.

        If i=1 then
         $\mathbf{v}_i = \mathbf{X}^T\mathbf{t}_i$, orthogonal loadings for deflation of $\mathbf{S}$
        Else
         $\mathbf{v}_i = \mathbf{X}^T\mathbf{t}_i - \mathbf{V}(\mathbf{V}^T\mathbf{X}^T\mathbf{t}_i))$, make $\mathbf{v}_i$ orthogonal to previous loadings.
        End
        $\mathbf{v}_i = \mathbf{v}_i/sqrt(\mathbf{v}_i^T\mathbf{v}_i)$, normalize loadings.
        $\mathbf{S}_{i+1} = \mathbf{S}_i - \mathbf{v}_i(\mathbf{v}_i^T\mathbf{S}_i)$, deflate $\mathbf{S}$ with respect to current loadings.
        Store $\mathbf{c}_i, \mathbf{w}_i, \mathbf{t}_i, \mathbf{u}_i, \mathbf{p}_i, \mathbf{q}_i, \mathbf{v}_i$ into $\mathbf{C}, \mathbf{W}, \mathbf{T}, \mathbf{U}, \mathbf{P}, \mathbf{Q}, \mathbf{V}$ respectively.
    End for

OUTPUT:
    $\mathbf{B} \leftarrow \mathbf{W}\mathbf{C}^T$, regression coefficients.
```

Figure 2.4: **Pseudocode for SIMPLS.**

2.1.3.5 Determining the number of PLS latent factors

One step of PLS regression analysis is to decide the optimal choice of number of latent factors K. The maximal value of K is the rank of matrix of predictors **X**: $K_{max} = rank(\mathbf{X})$. If K_{max} is chosen, then PLS will have the same number of components as ordinary multiple regression. However, an important feature of PLS regression is that usually much fewer factors are required to replace original predictors. Thus we would like to choose a small value of K but without sacrificing too much predictive power. One approach is to construct the PLS predictive model for a given number of factors on one set of data and then test it on another. We then choose the number of extracted factors such that the prediction error on the test data is minimal. Specifically, we use the standard procedure of 10-fold cross-validation to determine the minimum value of K. Cross-validation is used to evaluate machine learning algorithms as follows: the data is first partitioned into 10 equally sized sets or folds. Then 10 iterations of training and validation are performed and within each iteration, 9 folds of data are used for learning and the learned model makes predictions on the data in the remaining validation fold. Specifically for PLS, we have the following procedure:

1. Split all examples into training set consisting of 9 folds of data and test set containing the remaining examples.

2. Use the training set to determine the matrix of regression coefficients **B** for different values $K = 1, ..., K_{max}$.

3. Predict the responses of examples from the test set using **B** with different values of K.

4. Repeat step 1-3 10 times and compute the normalized mean squared error (NMSE) for every K.

5. Finally the value of K giving the smallest normalized mean squared error is selected.

2.1.4 Boosting

2.1.4.1 Introduction to the Idea of Boosting

Boosting is one of the most important developments in classification methodology. Kearns and Valiant first introduced the framework for Probably Approximately Correct (PAC) learning Kearns and Valiant (1988). The PAC-learning framework explores the limitations of existing machine learning algorithms. The work introduced the concept of weak learners versus strong learners: strong learner has good prediction performance with very low generalization error in a classification setting; while weak learner yields generalization error that is just lower than 12, which is only slightly better than random guessing.

In the PAC-learning framework, an instance space X is defined for the learning problem. We assume some distribution exists for X and we can sample from X based on this distribution. A concept c in X is some rule associated with a subset of X. And a concept class C is then defined as all possible instances of a given set of rules. The concept class C is PAC learnable if there exists an algorithm L with the following property: for any concept $c \in C$, any distribution on X and inputs $0 < \epsilon < \frac{1}{2}$ and $0 < \delta < 1$, L produces a hypothesis h with a generalization error smaller than ϵ with probability higher than δ. The algorithm L is called weak learnable if L has a hypothesis h for the concept c with probability δ and error rate ϵ as follows:

$$\begin{aligned} \delta &= \frac{1}{q(n, size(c))} \\ \epsilon &\leq \frac{1}{2} - \frac{1}{p(n, size(c))} \end{aligned}$$

where n is dimension of space X and p, q are polynomial functions. One intuitive interpretation of weak learner is that instead of producing a strong learner having arbitrary accuracy with arbitrary confidence, weak learner only guarantees a classifier with a fixed confidence that has accuracy slightly greater than $\frac{1}{2}$.

The question is whether we can "convert" a weak learner into a strong learner. Schapire Schapire (1990) later showed that there were techniques to improve the accuracy and confidence so that any weak learner can be "boosted" into a strong learner by averaging multiple weak hypotheses into a strong classifier. These techniques are referred to as "boosting" methods and the weak learner which only predicts better than chance was "boosted" into a strong learner with very good prediction accuracy. Figure 2.1.4.1 illustrates how boosting uses a weighted average of simple weak learners y_i to construct a strong one learner Y.

2.1.4.2 AdaBoost

The most popular implementation of boosting is adaptive boosting called AdaBoost proposed by Freund and Schapire Freund and Schapire (1997); Freund et al. (1999). AdaBoost is the first practical and efficient machine learning algorithm for ensemble learning. The idea of AdaBoost is to constructs an additive combination of weak rules by minimizing an exponential loss function.

In the boosting setting, we have a set of labeled examples $(x_i, y_i)_{i=1,\ldots,m}$ in $X \times Y$. x_i belongs to the instance or feature space X and label y_i belongs to the label space Y. Generally we assume $Y = \{-1, +1\}$ for binary classification problem. Suppose we have weak rules h: $X \rightarrow \{0, +1\}$. Though generally the output of h is assumed to lie in $\{-1, +1\}$, we will restrict to $\{0, +1\}$, which is the case for our study. In other words, weak rule can make a prediction that labels a given example, and it is also allowed to "abstain" by predicting 0, or say "I don't know" for certain examples. AdaBoost repeatedly calls a weak rule for a number of iterations $t = 1 \ldots T$ and linearly combines those moderately inaccurate weak

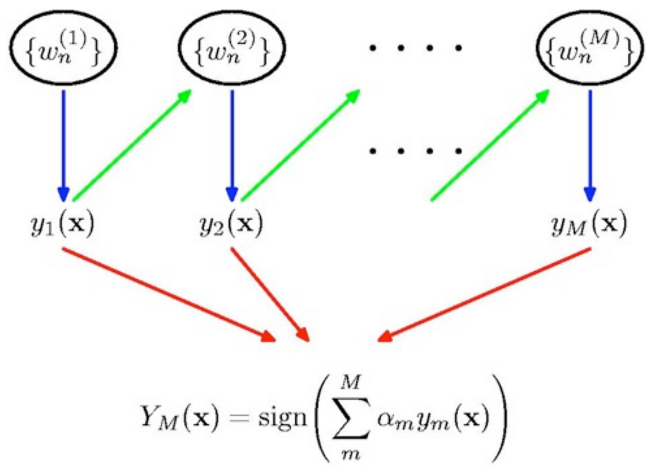


Figure 2.5: **Schematic illustration of Boosting Method.**

rules h^t into a single strong prediction function.

$$f(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{2.13}$$

The final prediction function $f(x)$ is a weighted majority vote of all T weak rules. We interpret the sign of $f(x)$ as the final classification for example x (-1 or +1) and the absolute $|h(x)|$ as the level of "confidence" of this classification. If $h(x)$ is far away from zero, it suggests a high confidence prediction. α_t is computed by the boosting algorithm as the weight assigned to each weak rule, measuring the contribution of weak rules in the final prediction. Since α_t can be either positive or negative, each weak rule can classify examples into both +1 and -1 classes even if h_t outputs $\{0, +1\}$. All α_t are initially set to zero. Then at every boosting iteration, AdaBoost selects a weak rule h_t and updates coefficient α_t.

The boosting algorithm also maintains a weight distribution over the training examples denoted by w_i (i=1,...,m). Initially all of the weights are set equally $w_i = \frac{1}{m}$ so the choice of the first weak rule has no bias towards any set of training examples. For later iterations $t = 2, ..., T$, weights of examples are modified and the boosting algorithm is then applied to the new weighted distribution. At each iteration t, examples misclassified by previous weak rule h_{t-1} are assigned higher weights, while correctly classified examples have lower weights. By doing this, AdaBoost forces each new weak rule to concentrate on the hard examples that are missed by previous one.

It has been shown that AdaBoost minimizes an exponential loss function to learn weak rules h_t and update α_t, w_i Hastie et al. (2001):

$$\begin{aligned} L &= \sum_{i=1}^{m} exp(-y_i f(x_i)) \\ &= \sum_{i=1}^{m} exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)) \end{aligned} \tag{2.14}$$

At iteration t-1, we write the strong prediction function as

$$f_{t-1}(x) = \sum_{q=1}^{t-1} \alpha_q h_q(x) \tag{2.15}$$

Using the exponential loss function, we solve

$$\begin{aligned}
(h_t, \alpha_t) &= argmin_{h,\alpha} L \\
&= argmin_{h,\alpha} \sum_{i=1}^{m} exp(-y_i(f_{t-1}(x) + \alpha h(x_i))) \\
&= argmin_{h,\alpha} \sum_{i=1}^{m} exp(-y_i \sum_{q=1}^{t-1} \alpha_q h_q(x_i)) exp(-y_i \alpha h(x_i))) \qquad (2.16)
\end{aligned}$$

for the new weak rule h_t and corresponding coefficient α_t. The above equation can be expressed as

$$(h_t, \alpha_t) = argmin_{h,\alpha} \sum_{i=1}^{m} w_i exp(-y_i \alpha h(x_i)) \qquad (2.17)$$

with $w_i = exp(-y_i \sum_{q=1}^{t-1} \alpha_q h_q(x_i))$. Given the fact that h_t is restricted to $\{0, +1\}$, we define $W_0^{h_t(x_i)=0}$, $W_+^{h_t(x_i)=1}$ and $W_-^{h_t(x_i)=1}$

$$\begin{aligned}
W_0^{h_t(x_i)=0} &= \sum_{i:h_t(x_i)=0} w_i \\
W_+^{h_t(x_i)=1} &= \sum_{i:h_t(x_i)=1, y_i=1} w_i \\
W_-^{h_t(x_i)=1} &= \sum_{i:h_t(x_i)=1, y_i=-1} w_i \qquad (2.18)
\end{aligned}$$

Thus the loss function in equation 2.16 can be simplified as

$$L = W_0^{h_t(x_i)=0} + W_+^{h_t(x_i)=1} e^{-\alpha_t} + W_-^{h_t(x_i)=1} e^{\alpha_t} \qquad (2.19)$$

We set $\frac{dL}{d\alpha_t} = 0$ to solve for α_t

$$\alpha_t = \frac{1}{2} ln \frac{W_+^{h_t(x_i)=1}}{W_-^{h_t(x_i)=1}} \qquad (2.20)$$

Plugging α_t into equation 2.19, we have

$$h_t = argmin_h W_0^{h(x_i)=0} + 2\sqrt{W_+^{h(x_i)=1} W_-^{h(x_i)=1}} \qquad (2.21)$$

Therefore equation 2.21 describes how we choose new weak rule h_t at boosting iteration t. And we add it to the prediction function $f(x)$ with a coefficient given in equation 2.20. Hence it is obvious that AdaBoost takes a greedy gradient descent approach to learning and combining weak rules, because it optimizes the loss function iteratively by adding one weak rule and updating its corresponding coefficient at a time. Finally for the next iteration t+1 the example weights are updated

$$w_i \propto w_i exp(-\alpha_t y_i h_t(x_i)) \tag{2.22}$$

So the new weights are $w_i = exp(-y_i \sum_{q=1}^{t} \alpha_q h_q(x_i))$. By doing this, the boosting algorithm increases weights for misclassified examples $y_i h_t(x_i) = -1$ and decreases weights for correctly classified ones $y_i h_t(x_i) = 1$. Figure 2.6 shows the pseudo-code for the AdaBoost algorithm. In addition, it has been shown that given $\frac{dL}{d\alpha_t} = 0$, the updated weight distribution is now "decorrelated" with the $yh_t(x)$:

$$\sum_{i=1}^{m} y_i h_t(x_i) w_i = 0 \tag{2.23}$$

The advantage of a boosting algorithm is that it does not need to find a strong classifier at a time, but instead it performs an easier task: find a group of weak rules that are only slightly better than random guessing. Suppose we have training error ϵ_t of the weak rule h_t $\frac{1}{2} - \gamma_t$ $(0 < \gamma_t < \frac{1}{2})$. γ_t tells how much better is h_t than random prediction. Freund and Schapire (1995) proved that the training error had the following upper bound:

$$\prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)} = \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq exp(-2\sum_{t=1}^{T} \gamma_t^2) \tag{2.24}$$

```
INPUT:
    Training examples: $(x_1, y_1)$, . . ., $(x_m, y_m)$
    Initialized example weights: $w_i = \frac{1}{m}$, i=1,2,...,m
    set $H$ of possible weak rules $h$: $X \rightarrow \{0,1\}$
Algorithm:

    For t = 1 to T
        (a) Train the weak rule $h_t$ using weights $w_i$
            $h_t = argmin_{h \in H} W_0^{h(x_i)=0} + 2\sqrt{W_+^{h(x_i)=1} W_-^{h(x_i)=1}}$
        (b) Compute $\alpha_t$
            $\alpha_t = \frac{1}{2} ln \frac{W_+^{h_t(x_i)=1}}{W_-^{h_t(x_i)=1}}$
        (c) Update $f(x_i) \leftarrow f(x_i) + \alpha_t h_t(x_i)$
        (d) Update $w_i \leftarrow w_i exp(-\alpha_t y_i h_t(x_i))$
    End for

OUTPUT:
    Final classifier is a weighted majority vote of the T weak rules
    $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$
```

Figure 2.6: **Pseudocode for AdaBoost.**

If each weak rule is slightly better than random, the training error drops exponentially fast. Earlier boosting algorithms require that a low bound γ is known $\gamma_t > \gamma$ before the boosting run. However, in practice it is not easy to know γ in advance. On the other hand, AdaBoost does not need to obtain this knowledge because of its"adaptive" nature: it "adapts" to the performance of each new weak rule at every boosting iteration. For example, α_t is solely determined by one weak rule h_t only. This is why the boosting algorithm is called AdaBoost – "Ada" is short for "Adaptive".

Early experiments found that AdaBoost did not overfit Breiman (1998); Drucker and Cortes (1996). The test loss, which is the estimate of generalization error, decreases even after thousands of boosting runs. It was sometimes observed that AdaBoost continued decreasing the test error even after the training error reached zero. Later Schapire et al. (1998a) gave an explanation of these empirical observations in terms of *margins* of training examples and showed that the increasing margin was related to decreasing generalization error. The margin of example (x_i, y_i) is defined as $y_i f(x_i)$, which is positive if f correctly classifies the example. The magnitude of the margin measures confidence in prediction and larger *margin* translates into a lower uppder bound on lower generalization error Freund et al. (1999). Even AdaBoost does not directly optimize margin, it is empirically observed that AdaBoost could aggressively improve margin. Because AdaBoost assigns more weight $w_i = exp(-y_i f(x_i))$ to examples with lower margin $y_i f(x_i)$ so it iteratively selects weak rules that concentrate on predicting those low margin examples.

2.1.4.3 Alternating Decision Trees

Boosting algorithms have used either decision stumps or decision trees as weak rules. For example, boosting decision stumps has T decision stumps and then takes a weighted majority vote of those stumps for the final classification. This assumes no relationship between weak rules, giving an unstructured set of T weak rules. Decision trees adds structure to the set of weak rules by building new weak rules based upon earlier ones. This makes it easier than boosting stumps to infer correlations between features. The boosting procedure has been successfully combined with decision tree algorithms to produce accurate classifiers such as CART (Classification and regression tree) and C4.5 Friedman et al. (1986); Freund and Schapire (1996); Quinlan (1996). Here we use the AdaBoost algorithm in the form of *alternating decision trees* (ADTs) Freund and Mason (1999b), which is a generalization of decision trees. An alternating decision tree consists of both *splitter nodes* and *prediction nodes*. Splitter nodes specify a condition associated with weak rule and prediction nodes contain a single number. Figure 2.1.4.3 shows an example of alternating decision tree. The tree starts with a root prediction node related to ratio of positive to negative training examples. At each boosting iteration a new splitter node (rectangle) and its prediction node (oval) containing the coefficient α_t are introduced. The splitter node is associated with the new weak rule h_t and asks question like "is $h_t(x) = 1$?". Only the examples x_i for which h_t predicts 1 will have α_t in the prediction node added to their prediction function $f(x_i)$. Moreover, only those examples will go further down in the tree through other splitter nodes below the current one. Therefore at each boosting iteration a new weak rule only makes predictions on the subset of the training examples that reach the splitter node.

In an alternating decision tree, one prediction node can be followed by multiple splitter nodes. For example in figure 2.1.4.3, the prediction node of h_1 has the splitter nodes of h_3 and h_4 attached. Therefore an example is classified by following all paths for which all splitter nodes are true and summing any prediction nodes connected. This is different from traditional decision trees such as CART or C4.5 in which an example follows only one path in the tree. A pseudo-code description of the alternating decision tree algorithm is given in Figure 2.7.

2.1.4.4 LPBoost

Previous study has shown that AdaBoost is very successful boosting algorithm because of its simplicity and high classification accuracy. And it has been empirically observed that AdaBoost rarely overfits even after the training error reaches zero. This observation has been explained in terms of margins of the training set, where the margin is interpreted as the confidence in the prediction Schapire (1999). AdaBoost is empirically found to improve the margins even after many boosting iterations, which are translated into better performance on the test set. Following the logic of the margins theory, alternative boosting algorithms have been proposed

```
INPUT:
  Training examples: $(x_1, y_1), . . ., (x_m, y_m)$
  Initialized example weights: $w_i = \frac{1}{m}$, i=1,2,...,m
  Initial set $H_0$ of possible weak rules $h$: $X \to \{0,1\}$

Algorithm:

  For t = 1 to T
    (a) Train the weak rule $h_t$ using weights $w_i$
        $H = \cup_{i=0}^{t-1} H_i$
        $h_t = argmin_{h \in H} W_0^{h(x_i)=0} + 2\sqrt{W_+^{h(x_i)=1} W_-^{h(x_i)=1}}$
    (b) Compute $\alpha_t$
        $\alpha_t = \frac{1}{2} ln \frac{W_+^{h_t(x_i)=1}}{W_-^{h_t(x_i)=1}}$
    (c) Update $f(x_i) \leftarrow f(x_i) + \alpha_t h_t(x_i)$
    (d) Update $w_i \leftarrow w_i exp(-\alpha_t y_i h_t(x_i))$
    (e) Update set $H_t$ of possible weak rules
        for every $h \in H_0$, define a new weak rule
        $h_{new}(x) = h(x)$, if $h_t(x) = 1$
                  $= 0$,    if $h_t(x) = 0$
        Define $H_t$ as the set of all new weak rules $h_{new}$
  End for

OUTPUT:
  Final classifier is a weighted majority vote of the T weak rules
  $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$
```

Figure 2.7: **Pseudocode for the alternating decision tree algorithm Freund and Mason (1999a).**

that would provably optimize margins as AdaBoost is not theoretically perfect from the viewpoint of optimization as the training is an iterative gradient-descent procedure. LPBoost, linear programming (LP) based boosting, is one of them Grove and Schuurmans (1998); Demiriz et al. (2002). Different from the AdaBoost, LPBoost produces the optimal linear combination of weak rules by optimizing a margin cost function.

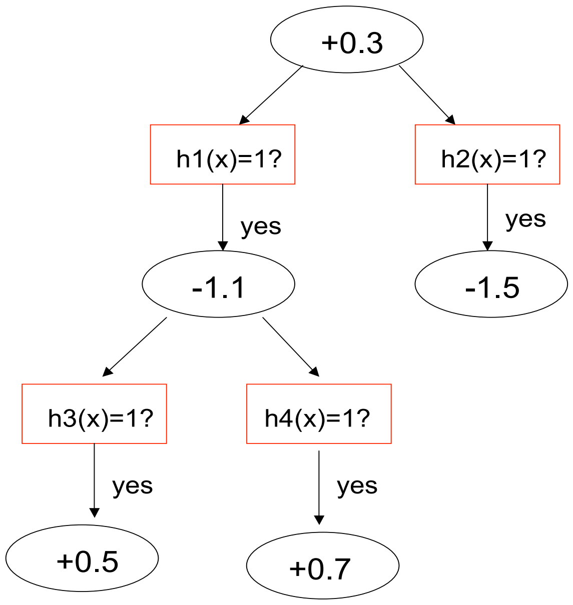


Figure 2.8: **Example of an alternating decision tree.**

Let the matrix H be a t by m matrix of all the possible labelings of the training examples, where t is number of weak rules and m number of training examples. Specifically $H(x_i) = [h_1(x_i), ..., h_t(x_i)]$ is vector of labels given by weak rule $h_1, ..., h_t$ on the training example x_i. After a new weak rule h_t is received, the margin of an example (x_i, y_i) measures how well the example is classified by the current combination of weak rules and is defined as $\rho(\alpha, x_i) = y_i \sum_{q=1}^{t} \alpha_q h_q(x_i)$. When the sign of the combination is the same as the label y_i, the margin is positive. The margin of a set of examples is always the minimum margin of the set. One measure of the performance of weak rule h_q with respect to weight distribution w is "edge", which is defined as $\gamma(w, h_q) = \sum_{i=1}^{m} y_i h_q(x_i) w_i$. The edge indicates how well a weak rule classifies the training examples. The higher the edge, the higher classification accuracy the weak rule has. The edge of a set of weak rules is defined as the maximum edge of the set. We ask the question: how are margins and edges related? By linear programming duality, the minimum edge of the set of weak rules equals the maximum margin of the set of examples:

$$\begin{aligned} Minimum\ edge : \gamma_t^* &= min_w max_{q=1,\ldots,t} \gamma(w, h_q) \\ &= min_w max_{q=1,\ldots,t} \sum_{i=1}^{m} y_i h_q(x_i) w_i \\ Maximum\ margin : \rho_t^* &= max_\alpha min_{i=1,\ldots,m} \rho(\alpha, x_i) \\ &= max_\alpha min_{i=1,\ldots,m} \sum_{q=1}^{t} \alpha_q h_q(x_i) y_i \\ Duality\ \gamma_t^* &= \rho_t^* \end{aligned} \tag{2.25}$$

Equation 2.25 results in the hard margin version of LPBoost Grove and Schuurmans (1998). The hard margin version of LPBoost solves for $\alpha_1, ..., \alpha_t$ by maximizing the minimum margin over all examples. And the dual variables of the linear program provide the updated example weights needed by the boosting algorithm $w_1, ...w_m$. If the training examples can be separated by the linear combination of weak rules, maximizing the margin has been proved to be a proxy for low generalization error Schapire et al. (1998b). If the data is not separable, we replace the hard margin by the soft margin. By "soft" we mean relaxing the lower bound on margin. Training examples are allowed to lie below the margin but they are penalized through slack variables. To maximize the soft margin of training examples Demiriz et al. (2002), LPBoost solves the following LP problem mathematically :

$$\begin{aligned} max_{\alpha,\xi,\rho} \quad & \rho - D\sum_{i=1}^{m}\xi_i \\ s.t. \quad & y_i H(x_i)\boldsymbol{\alpha} + \xi_i \geq \rho \\ & \xi_i \geq 0, i = 1\dots m \\ & \sum_{q=1}^{t}\alpha_q = 1, \alpha_q \geq 0, q = 1, ..., t \end{aligned} \tag{2.26}$$

This LP formulation is for soft-margin LPBoost, which is similar to the l_1-norm SVM. Here $\rho - D\sum_{ge}\xi_{ge}$ is the soft margin to be optimized at boosting iteration t. ξ_i is the slack variable for each training example. D is the tradeoff parameter that balances margin maximization and training error. This LP formulation is also known as ν-LP is with $D = \frac{1}{N\nu}(\frac{1}{N} \leq \nu \leq 1)$, as ν is interpreted as percentage of training examples misclassified Ratsch et al. (1999). In experiments, D is chosen by a cross-validation procedure. Without D and slack variables ξ, it is the hard-margin version of LPBoost Grove and Schuurmans (1998).

The dual of the LP problem is:

$$\begin{aligned} min_{w^{t+1},\gamma} \quad & \gamma \\ s.t. \quad & \sum_{i} y_i h_q(x_i) w_i^{t+1} \leq \gamma \quad q = 1\dots t \\ & \sum_{ge} w_i^{t+1} = 1, 0 \leq w_i^{t+1} \leq D \end{aligned} \tag{2.27}$$

The dual LP problem defines a basic boosting algorithm: update w^{t+1} such that the maximum edge of the t weak rules received so far is minimized. Specifically, we have a constraint that h^q achieves edge of at most γ, which is minimized.

The dual LP has a natural interpretation. w_i^{t+1} is viewed as a probability distribution over the training examples x_i that sum to 1. $\sum_{i=1}^{m} y_i h_q(x_i) w_i^{t+1}$ is simply the weighted sum of the correctly classified training examples minus the weighted sum of misclassified training examples. The set of weak rules satisfy the constraint $\sum_i y_i h_q(x_i) w_i^{t+1} \leq \gamma \quad (q = 1 \ldots t)$ so γ is the maximum edge of the set. The objective of dual LP is to find a reweighting of the training set, such that maximum edge γ of the set of weak rules is as small as possible. Therefore LPBoost is "totally corrective" in the sense that it optimizes the weights based on all past weak rules. In contrast, AdaBoost only updates the weights based on the last weak rule. The pseudocode description for LPBoost combined with alternating decision tree is given in Figure 2.9.

In a linear program a column corresponds to a primal variable or a weak rule in LPBoost . In many circumstances the size of the set of weak rules is too large to use standard LP solvers. Because the matrix H has a very large number of columns and so the primal LP has extremely more variables than constraints. Therefore the idea of solving the LPBoost formulation could be intractable using standard LP techniques. To resolve this issue, the classic column generation (CG) based simplex algorithm Luenberger and Ye (2008) has been proposed and applied to the LP Bennett et al. (2000); Demiriz et al. (2002). Column generation is a state-of-the-art method for solving difficult large-scale optimization problems. Unlike traditional LP solvers, the CG method avoids considering all variables of an optimization problem explicitly. Thus it is very powerful for LP with significantly more variables than constraints.

The logic of CG is: the number of non-zero variables of the optimal solution is equal to the number of constraints, therefore even though the number of possible variables may be large, we only need a small subset of the entire variable set, i.e., a subset $\hat{H}$ of the columns of H to learn the current solution. The LP solved using

$\hat{H}$ is known as the *restricted master problem.* Solving the restricted primal problem also leads to solving a relaxation of the dual LP since the constraints for weak rules that have not be generated as columns are not taken into account.

The CG method first solves the restricted master problem and then asks the question:" Are there any other variables that could be included to improve the objective function?". If it is determined that the current solution is not optimal, the algorithm will continue to generate a new column that will improve the solution. This creates a subproblem to identify a new variable. The objective function of the subproblem is the reduced cost of the new variable with respect to the current dual variables. Thus given the current set of dual values, one either identifies a variable that has a negative reduced cost or finds that no such variable exists. If it is the former, this variable is then added to get the new restricted master problem, and the new restricted master problem is re-solved. It will generate a new set of dual variables, and the subproblem is re-solved. This process is repeated until no negative reduced cost variables are identified and then we can conclude that the solution to the master problem is optimal. In summary, CG finds the variables with negative reduced costs without going through all variables explicitly.

2.1.4.5 Connection between AdaBoost and LPBoost

LPBoost is mainly inspired by SVM and the idea of LPBoost is to maximize the minimum margin over the training set because it is believed that the minimum margin is an effective proxy for generalization error Schapire et al. (1998b). The question is: is there any connection between AdaBoost and LPBoost? It can be shown that the hard margin version of LPBoost actually minimizes an upper bound of AdaBoost's cost function.

```
INPUT:
  Training examples: $(x_1, y_1), . . ., (x_m, y_m)$
  Initialized example weights: $w_i = \frac{1}{m}$, i=1,2,...,m
  Initial set $H_0$ of possible weak rules $h$: $X \rightarrow \{0,1\}$
  Soft margin parameter $\nu$
  Penalization constant $D = \frac{1}{m\nu}$

Algorithm:
  For t = 1 to T
    (a) Train the weak rule $h_t$ using weights $w_i$
        $H = \cup_{i=0}^{t-1} H_i$
        $h_t = argmin_{h \in H} W_0^{h(x_i)=0} + 2\sqrt{W_+^{h(x_i)=1} W_-^{h(x_i)=1}}$
    (b) Optimize weights of weak rules $\alpha_q, q = 1, ..., t$
            $max_{\alpha,\xi,\rho}$   $\rho - D\sum_{i=1}^{m} \xi_i$
            s.t.          $y_i \sum_{q=1}^{t} \alpha_q h_q(x_i) + \xi_i \geq \rho \quad i = 1 \ldots m$
                          $\sum_{q=1}^{t} \alpha_q = 1, \xi_i \geq 0$
                          $\alpha_q \geq 0, q = 1, ..., t$
    (c) Update $f(x_i) = \sum_{q=1}^{t} \alpha_q h_q(x_i)$
    (d) Update $w_i^{t+1}$ by solving the dual LP
              $min_{w^{t+1},\beta}$   $\beta$
              s.t.          $\sum_{i=1}^{m} y_i h_q(x_i) w_i^{t+1} \leq \beta \quad q = 1 \ldots t$
                            $\sum_{i=1}^{m} w_i^{t+1} = 1,\ 0 \leq w_i^{t+1} \leq D$
    (e) Update set $H_t$ of possible weak rules
        for every $h \in H_0$, define a new weak rule
        $h_{new}(x) = h(x)$, if $h_t(x) = 1$
                  $= 0$,   if $h_t(x) = 0$
        Define $H_t$ as the set of all new weak rules $h_{new}$
  End for

OUTPUT:
  Final classifier is a weighted majority vote of the T weak rules
  $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$
```

Figure 2.9: **Pseudocode for LPBoost combined with alternating decision tree algorithm.**

Let us take a look at the cost function of AdaBoost and LPBoost in the primal. We know that AdaBoost minimizes the exponential cost function in terms of margins of examples:

$$minimize \sum_{i=1}^{m} exp(-y_i \sum_{t=1}^{T} \alpha_t h_t(x_i)) = min \sum_{i=1}^{m} exp(-margin(x_i, y_i)) \qquad (2.28)$$

which is the same as

$$minimize\ log(\sum_{i=1}^{m} exp(-margin(x_i, y_i))) \qquad (2.29)$$

So AdaBoost minimizes a log-sum-exp cost function, which can be viewed as a smooth approximation of the maximum function. We have the following inequality:

$$max_i\ a_i \leq log(\sum_{i=1}^{m} exp(a_i)) \leq max_i\ a_i + log(m) \qquad (2.30)$$

If we apply the inequality 2.30 to equation 2.29, we can show that the margin maximization problem of LPBoost is equivalent to minimizing the bound on AdaBoost's cost function given the following:

$$maximize\ min_i\{margin(x_i, y_i)\} = minimize\ max_i\ \{-margin(x_i, y_i)\} \quad (2.31)$$

Therefore, LPBoost uses a hard maximum function while AdaBoost uses a soft approximation of the maximum function.

2.1.4.6 TotalBoost

LPBoost is the most straightforward boosting algorithm for maximizing the soft margin of the linear combination of weak rules. It does so by solving a linear programming problem. In the dual LP, the boosting algorithm minimizes edge γ without placing any specific constraints on example weights. One challenge of this is that the weights computed by the CG simplex method are often sparse, i.e. $w_i = 0$ for many examples Bennett et al. (2000); Demiriz et al. (2002); Warmuth et al. (2006). The dual LP has a basic feasible solution corresponding to a vertex of the dual feasible region. While a face of the region corresponding to many nonnegative weights may be optimal, a simplex method is more likely to choose the vertex solution with fewer nonnegative weights. Therefore with examples having zero weights $w_i = 0$, solutions based on small data subsets are not meaningful for the entire training set, In other words, it could easily happen that LPBoost is "blind" on a large subset of training examples when selecting new weak rules.

In this chapter, we discuss another totally corrective boosting algorithm TotalBoost which also maximizes the margin. TotalBoost is "totally corrective" because the algorithm also optimizes weights on examples based on all past weak rules Warmuth et al. (2006).

However, different from LPBoost, TotalBoost attempts to maintain a stable weight distribution. To do this, TotalBoost introduces an entropic regularization on the weights that helps to improve the stability of the boosting procedure. LPBoost updates example weights by minimizing the maximum edge of all past weak rules, while the basic idea of TotalBoost is to constrain the edges of all past weak rules only and minimize the relative entropy to the initial weight distribution. In this aspect, TotalBoost has its connection with AdaBoost. Because AdaBoost can be viewed as minimizing the relative entropy to the last weight distribution subject to the constraint that the edge of the last weak rule is zero Kivinen and Warmuth (1999).

The minimum relative entropy principle works as follows: among all the weight solutions that satisfy linear constraint on edge γ, it chooses the new weight distribution w^{t+1} that is 'closest' to the initial distribution w^0. The 'closeness' is measured by the relative entropy between distribution w and w^0 defined as follows: $\triangle(w, w^0) = \sum_{i=1}^{m} w_i \log \frac{w_i}{w_i^0}$. The optimization problem is defined in the following equation , where TotalBoost minimizes the relative entropy subject to the constraint that edges of all past weak rules are less than or equal to zero.

$$\begin{aligned} w^{t+1} = \quad & argmin_w \triangle(w, w^0) = argmin_w \sum_{i=1}^{m} w_i \log \frac{w_i}{w_i^0} \\ s.t. \quad & \sum_i w_i y_i h_q(x_i) \leq 0 \quad q = 1 \dots t \\ & \sum_i w_i = 1 \quad w_i \geq 0 \end{aligned} \tag{2.32}$$

In LPBoost, the upper bound γ on the edge is chosen to be as small as possible, whereas in TotalBoost the weights are chosen to be closest to initial weights as long as γ is less than zero. Therefore in TotalBoost γ decreases more moderately and TotalBoost could converge slowly than LPBoost during early boosting rounds. However, relative entropy minimization has a smoothing effect in the weight distribution, as it is "slowly" changed from the initial distribution . Since the initial weight distribution w^0 is uniform, it is unlikely that TotalBoost produces problematic sparse weight distribution during later boosting iterations. In addition, the relative entropy term makes the objective function of TotalBoost strictly convex and therefore the optimization problem has a unique solution, which we denote as w^{t+1}. In contrast, the objective function of LPBoost is linear so the solution is not unique and highly dependent on the LP solver used.

A pseudocode description for TotalBoost combined with alternating decision tree algorithm is given in Figure 2.10. We could use a "vanilla" *sequential quadratic programming* (SQP) algorithm Nocedal and Wright (1999) to solve the convex optimization problem in TotalBoost.

$$\min_{w:\sum_i w_i=1, w_i\geq 0, \sum_i w_i y_i h_q(x_i)\leq 0, q=1\dots t} \sum_{i=1}^{m} w_i \log \frac{w_i}{w_i^0}$$

We first set the approximate solution to $\hat{w} = w^0$ and optimize $\hat{w}$ in a sequential way. Given the current solution $\hat{w}$ that satisfies the constraints $\sum_i \hat{w}_i =$

$1, \hat{w}_i \geq 0$, we determine the update w by minimizing the approximation of change in relative entropy $\triangle(w, w^0) - \triangle(\hat{w}, w^0) = \sum_{i=1}^m \frac{1}{2\hat{w}_i}((w_i - \hat{w}_i)^2 + 2\hat{w}_i \log \frac{\hat{w}_i}{w_i^0}(w_i - \hat{w}_i))$ subject to the same linear constraints. The estimate $\hat{w}$ is then updated $\hat{w} \leftarrow w$ and this process is repeated until $\hat{w}$ converges to a distribution.

$$\boxed{\begin{aligned} min_w \quad & \sum_{i=1}^m \frac{1}{2\hat{w}_i}((w_i - \hat{w}_i)^2 + 2\hat{w}_i \log \frac{\hat{w}_i}{w_i^0}(w_i - \hat{w}_i)) \\ s.t. \quad & \sum_{i=1}^m w_i y_i h_q(x_i) \leq 0 \quad q = 1 \dots t \\ & \sum_{i=1}^m w_i = 1 \end{aligned} \qquad (2.33)}$$

The above objective function is the 2nd order of Taylor approximation of change in the relative entropy. Therefore the relative entropy optimization problem is broken down into a series of quadratic optimization problems with a diagonal Hessian, which can be easily solved by off-the-shelf optimizers.

```
INPUT:
    Training examples: $(x_1, y_1), \ldots, (x_m, y_m)$
    Initialized example weights: $w_i = \frac{1}{m}$, i=1,2,...,m
    Initial set $H_0$ of possible weak rules $h$: $X \rightarrow \{0,1\}$
    Soft margin parameter $\nu$
    Penalization constant $D = \frac{1}{m\nu}$

Algorithm:
    For t = 1 to T
        (a) Train the weak rule $h_t$ using weights $w_i$
            $H = \cup_{i=0}^{t-1} H_i$
            $h_t = argmin_{h \in H} W_0^{h(x_i)=0} + 2\sqrt{W_+^{h(x_i)=1} W_-^{h(x_i)=1}}$
        (b) Optimize weights of weak rules $\alpha_q, q = 1, ..., t$
                $max_{\alpha,\xi,\rho}$   $\rho - D\sum_{i=1}^{m} \xi_i$
                s.t.          $y_i \sum_{q=1}^{t} \alpha_q h_q(x_i) + \xi_i \geq \rho \quad i = 1 \ldots m$
                              $\sum_{q=1}^{t} \alpha_q = 1, \xi_i \geq 0$
                              $\alpha_q \geq 0, q = 1, ..., t$
        (c) Update $f(x_i) = \sum_{q=1}^{t} \alpha_q h_q(x_i)$
        (d) Update $w_i^{t+1}$ by minimizing the relative entropy
                $min_{w^{t+1}}$   $\triangle(w^{t+1}, w^0)$
                s.t.          $\sum_{i=1}^{m} w_i^{t+1} y_i h_q(x_i) \leq 0 \quad q = 1 \ldots t$
                              $\sum_{i=1}^{m} w_i^{t+1} = 1$
        (e) Update set $H_t$ of possible weak rules
            for every $h \in H_0$, define a new weak rule
            $h_{new}(x) = h(x)$, if $h_t(x) = 1$
                     $= 0$,    if $h_t(x) = 0$
            Define $H_t$ as the set of all new weak rules $h_{new}$
    End for

OUTPUT:
    Final classifier is a weighted majority vote of the T weak rules
    $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$
```

Figure 2.10: **Pseudocode for TotalBoost combined with alternating decision tree algorithm.**

2.2 Biology of gene regulation

2.2.1 From DNA to Protein

Deoxyribonucleic acid (DNA), Ribonucleic acid (RNA) and proteins are the three major macromolecules that are essential to all living organisms. DNA contains the genetic instructions used in the development and function of the whole organism. For example, many DNA molecules store genetic information that are needed to build components of cells, e.g. RNA and protein molecules. DNA sequences that are eventually translated into proteins or converted into functional RNA molecules are called genes.

In 1953, two biologists Watson and Crick proposed the double helix structure for DNA Watson and Crick (2003). The units of a DNA molecule are nucleotides and there are four different types of nucleotides: adenine (A), guanine (G), cytosine (C) and thymine (T). Every nucleotide pair along the double helix is called a *base pair*(bp) and the DNA obeys the following complementary base pairing rule: each type of nucleotide on one strand normally connects with just one type of nucleotide on the other strand. Specifically, A bonds only to T and C only to G. Figure 2.11 illustrates the double-helix structure of DNA molecule.

RNA is created from DNA via a biological process called transcription. Like DNA, RNA is also made up of nucleotides. During transcription, a DNA sequence is read by an enzyme called RNA polymerase, which produces a complementary RNA copy of the DNA sequence. RNA copies the genetic information in DNA with the same nucleotides as DNA except for uracil (U), which plays the role of thymine (T) in DNA.

The DNA segment transcribed into a RNA molecule is called the coding sequence and encodes at least one gene. If the transcribed gene is eventually used to create the protein, the RNA is called messenger RNA (mRNA). If not, the

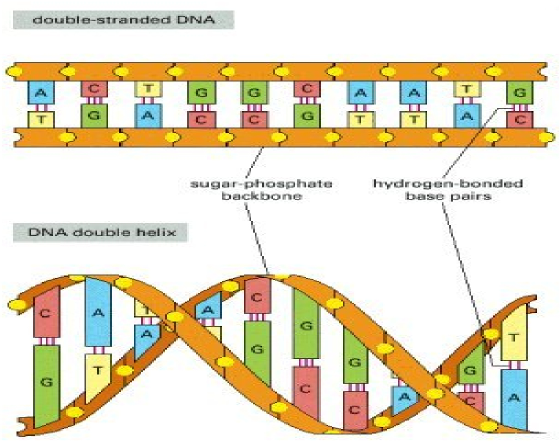


Figure 2.11: **DNA molecule has a double-helix structure.** The nucleotides A, C, G, T form complementary base pairs (AT) and (CG).

transcribed gene may encode for ribosomal RNA (rRNA), transfer RNA (tRNA) or other ribozymes.

mRNA carries information about a protein sequence. After the transcription process, the mRNA is exported from the nucleus to the cytoplasm and is then translated into its corresponding protein.

Proteins are made up of amino acids. During translation, the genetic code of mRNA relates the DNA sequence to the amino acid sequence in proteins. Every triplet of nucleotides in mRNA forms a codon and each codon corresponds to a particular amino acid. In this way, the mRNA sequence is used as a template to assemble a chain of amino acids that fold into 3 dimensional structures to form a protein.

2.2.2 Modeling of transcription factor binding sites

The part of DNA sequences transcribed into RNA molecules encode genes and is called coding region. But there are other DNA segments called non-coding region that regulate the use of the genetic information. RNA polymerase binds to the parts of non-coding region of an associated gene to perform the transcription of genetic information from DNA to RNA. A group of proteins called transcription factors also bind to specific DNA sequences in the non-coding region, which affects the rate of gene transcription. By doing this, transcription factors can either promote (as an activator), or block (as a repressor) the binding of RNA polymerase to in the transcription process. If the transcription factor is an activator, it increases the rate of transcription and thus expression level of the associated gene. If the transcription factor is a repressor, it decreases the rate of transcription. There are also proteins called coactivators or corepressors that work with transcription factors to increase or decrease transcription rate.

The non-coding region that transcription factors bind to is referred to as *promoter sequence*. In eukaryotes like *S. cerevisiae*, the core promoter for a gene transcribed is often found upstream of start site of the gene. A transcription factor regulating the transcription of a gene contains one or more DNA-binding domains (DBDs), which are attached to specific sequences of DNA in the promoter region of the associated gene. We can assign every transcription factor a set of potential binding sites, usually known as motifs, that the transcription factor binds to with a high probability. In this way, the promoter sequences provide information for which transcription factor controls the transcription of a gene.

A specific set of DNA binding sites associated with transcription factors, usually referred to as a binding site motif, can be represented by a consensus sequence. The consensus sequence contains new letters in the subsets of the A,C,G,T nucleotides. For example, we consider a short DNA sequence: C[GA]RN{T}. In this notation, C means C is always in the position; [GA] means either G or A; R stands for A or G; N means any nucleotide; {T} means any nucleotide but T. However, in this example, the notation [GA] does not give any information of probability of G or A occurrence.

To address this issue, the position specific scoring matrix (PSSM) is introduced to model the binding site motif in a probabilistic way. PSSM scores how close any sequence is to the set of sequences used to construct the scoring matrix. PSSM assumes independence between positions in the sequence, as it calculates scores at each position independently from the nucleotides at other positions. Therefore, the score of a sequence is simply the product of the scores or the sum of log of scores at each position. If the length of the binding site motif is l, a PSSM p is a matrix of size 4 by l, where each column j is a probability distribution over $\{A, C, G, T\}$: $\sum_{s \in \{A,C,G,T\}} p_j(s) = 1$. The probabilities $p_j(s_j)$ can be calculated by counting the nucleotides of each type at position j. The PSSM assigns every sequence $s_1 s_2 ... s_l$ a log-odds score $S = \sum_{j=1}^{l} log(p_j(s_j)/p_{bg}(s_j))$, where $p_j(s_j)$ is the probability of nucleotide s_j at position j in the PSSM p, and $p_{bg}(s_j)$ is the background probability of nucleotide s_j. The background probability of nucleotide can be viewed as a "prior" probability. For example, the background probability could be the frequency of the nucleotide in all sequences used to create the matrix. So given a sequence of length l the above log-odds score can be computed by summing up the log-odds score of every nucleotide s_j in the sequence.

2.2.3 Experimental Data

2.2.3.1 Microarray gene expression data

DNA microarrays measure mRNA levels in cells or tissues for many genes simultaneously. The goal of microarray experiments is to identify genes that are differentially transcribed under different biological conditions or in tissue samples. A microarray experiment has the following components: a set of probes, an array on which these probes are immobilised, a sample containing a mix of labeled biomolecules that bind to the probes. Single strands of complementary DNA for thousands of gene are immobilized in a microarray. From a sample of interest, the mRNA is extracted, labeled and hybridized to the microarray. The quantity of label of each spot can be measured and the measurement indicates the abundance level of corresponding RNA transcript, or expression level of the gene in the sample.

There are two common types of labeling. First, The company Affymetrix synthesizes sets of short oligomers for genes on a glass wafer and uses only one fluorescent label Lipshutz et al. (1999)(www.affymetrix.com). Each transcript has 16 to 20 pairs of oligonucleotide probes and each probe pair is an oligonucleotide of 25 bases that exactly matches the target sequence but only one mismatch in the middle. The purpose of the mismatch probes is to estimate background noise that contributes to the signal. The simple average or weighted average of probe intensities provide an estimate of the abundance of the transcript. Second, two samples are labeled, one with a green fluorescent dye (Cy3) and the other with a red dye (Cy5). The two samples are mixed and then both hybridized to the spots on the microarray, and the slide is scanned. From the scanned image, the intensities of Cy3 and Cy5 signals are read and the log of the ratio of Cy5 intensity to Cy3 intensity is calculated. The log ratio indicates relative level of gene expression in Cy5-labeled versus Cy3-labeled sample. Figure 2.12 gives an example of microarray data.

2.2.3.2 Gene ontology annotation data

In bioinformatics, one challenge in seeking biological information is that there is no standard terminology, e.g. inconsistent descriptions for the same gene exist in different biological databases. The Gene Ontology (GO) Ashburner et al. (2000) project was initiated to standardize the representation of gene and gene product properties in many different species. The ontology has three domains: cellular component, molecular function and biological process. The biological databases in the GO Consortium gather information about properties of genes using terms in the GO ontology, from published scientific papers. In this way, every gene in an organism can be annotated with a specific set of GO terms. In bioinformatics, research scientists study the enrichment of specific GO terms in gene sets, which helps to validate biological hypotheses made with computational algorithms.

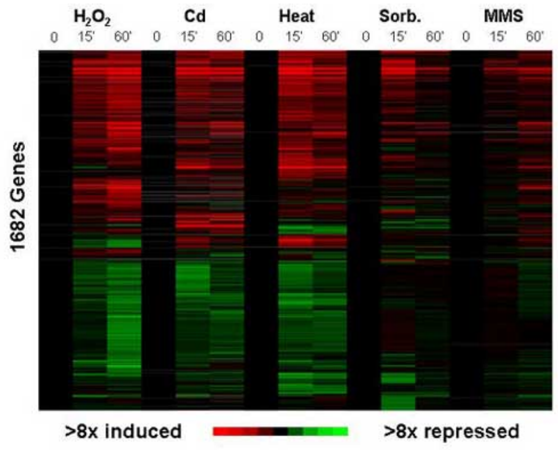


Figure 2.12: **Example of microarray gene expression data.** Every row is a gene and every column corresponds to a sample in a specific condition. Red color indicates over-expression of gene (induction) and green under-expression of gene (repression).

Chapter 3

Learning "graph-mer" motifs that predict gene expression trajectories

3.1 Introduction

The mRNA expression level of a gene is regulated by multiple input signals that are integrated by the *cis* regulatory logic encoded in the gene's promoter. genes whose regulatory sequences contain similar DNA motifs are likely to have correlated expression profiles across a given set of experimental conditions. The converse, however, is not necessarily true. That is, genes can have correlated expression profiles without being coregulated, since multiple regulatory programs may lead to similar patterns of differential expression. This is particularly evident in developmental time series data, in which the genes exhibit only a few distinct expression patterns. Nevertheless, computational approaches for deciphering gene regulatory networks from gene expression and promoter sequence data often do assume that correlation implies coregulation. For example, a typical computational strategy is to cluster genes by their expression profiles and then apply motif discovery algorithms to the promoter sequences for each cluster. The cluster-first motif discovery approach is indeed so prevalent that the best-known benchmarking study of motif discovery algorithms Tompa et al. (2005) defines the problem in precisely this way – namely, given a cluster of genes, find the overrepresented motif(s) in the promoter sequences – and compares numerous such algorithms. It is clear, however, that assigning genes to static clusters that are assumed to be coregulated oversimplifies the biology of transcriptional regulation. Moreover, in a setting where there are few experiments probing the conditions of interest or where many genes have synchronized expression profiles, such as in a time course, clustering may fail to resolve meaningful gene sets for subsequent motif analysis.

We present an algorithm that models the natural flow of information, from sequence to expression, to learn cis regulatory motifs and to characterize gene expression patterns. Our algorithm learns motifs that help to predict the full expression profiles of genes over a set of experiments, with no clustering. More precisely, we use a novel algorithm based on partial least squares (PLS) regression to learn a mapping from the set of k-mers in a promoter to the expression profile of the gene across experiments; in time series, we learn k-mers that help to predict the full expression time course for genes. PLS combines dimensionality reduction and regression; it iteratively finds latent factors in the input space with maximal covariance with projections in the output space. We introduce a graph-regularized version of the PLS algorithm to enable motif discovery by imposing two constraints: a lasso Tibshirani (1996) constraint for sparsity and a graph Laplacian constraint for smoothness over sequence-similar motifs.

The goal of our method is to discover regulatory elements and decipher transcriptional regulation in the nematode *Caenorhabditis elegans*, a key model organism in developmental biology. In particular, we are interested in using mRNA profiling experiments from developmental time courses, where the high global level of correlation presents a challenge to clustering. Dissection of gene regulatory logic is not as advanced in *C. elegans* as it is in *D. melanogaster*, for example. There are few motif discovery programs designed specifically for worms, and while worm biologists do use generic programs such as MEME Bailey and Elkan (1994a), traditionally they have relied on experimental strategies to define binding motifs and then performed genome-wide motif searches and validation with transgene reporters. One goal of our work is to advance this area of inquiry by defining novel elements and providing new opportunities for directed experimental validation.

Here, we evaluate the performance of our method by measuring normalized mean squared error on cross-validation test data. Moreover, we show that the learned PLS latent factors contain information that is both statistically significant and biologically meaningful. These significant features, which are associated with high generalization rather than simple correlations in the training data, suggest biological relationships between sequence motifs and temporal expression patterns. We first describe the data sets used in our study, and then present the algorithm to finally perform validation of learned PLS latent factors and motifs. Most of the work presented here is published in Li et al. (2010).

3.2 Data Sets

3.2.1 Microarray Data

We use the gene expression microarray data for wild-type germline development in worm *C. elegans* Reinke et al. (2004). This data set consists of a time course beginning in the middle of the third larval stage (L3) and extending through adulthood. During this time, the major developmental changes occur in the germ line. Some germ cells undergo constant proliferation, while others initiate developmental events, including entry into meiosis followed by differentiation into sperm, which occurs in the fourth larval stage, or differentiation into oocytes, which occurs in young adults. By the end of the timecourse, animals have produced mature gametes and launched embryogenesis. Twelve samples are collected at 3-hour intervals with 3 replicates for each sample. Basic microarray data normalization was performed in the original study, and we use the normalized gene expression levels as reported (Gene Expression Omnibus, http://www.ncbi.nlm.nih.gov/geo/, accession numbers GSE726-GSE737). We average expression levels over replicates for 20,000 genes and calculated the 5% and 95% quantile of all expression values. We filtere out genes with baseline expression (defined here as having expression values between the 5% and 95% quantiles at all time points) and also ones that exhibit little variance in expression over time ($SD < 0.1$). After further removing genes without upstream sequences from WormMart, we obtaine the gene expression matrix for ∼9,000 genes and 12 time points.

3.2.2 Promoter Sequences

We download promoter sequences spanning 500 bp upstream of transcription start sites from WormMart. For genes whose upstream intergenic sequence is shorter than 500 bps, we use the intergenic sequences instead of 500 bps upstream. We scan the promoter sequences for candidate 6-mers and 7-mers, and filter k-mers based on expected counts in background sequences.

3.3 Methods

Standard partial least squares regression

Since our algorithm builds on ideas from PLS regression, we first describe how to use standard PLS to iteratively learn a linear mapping from the promoter sequences of genes, as represented by their k-mer counts, and their mRNA expression profiles. Formally, using a training set of G genes, PLS takes a motif matrix $\mathbf{X}$ (dimension $G \times M$, where M is the number of k-mers), representing the individual k-mer counts for each gene, and a gene expression matrix by $\mathbf{Y}$ (dimension $G \times E$, where E is the number of experiments). Here, the columns of $\mathbf{X}$ represent the independent variables (features) and the columns of $\mathbf{Y}$ are the response variables; we also call $\mathbf{X}$ the input matrix and $\mathbf{Y}$ the output matrix. PLS then performs the following steps:

a. Scale $\mathbf{X}$ and $\mathbf{Y}$ so that each column of the input and output matrices has zero mean and unit variance.

b. Perform dimensionality reduction by construction of latent factors $\mathbf{T} = \mathbf{XW}$: Construct K *weight* vectors, placed as column vectors in $\mathbf{W}$ (dimension $M \times K$), and corresponding *latent* factors, placed as column vectors in $\mathbf{T}$ (dimension $G \times K$), where the weight vectors are chosen so that the latent factors have maximal covariance with directions in the multivariate response $\mathbf{Y}$.

c. Use the latent factors $\mathbf{T}$ to predict $\mathbf{Y}$: Regress $\mathbf{Y}$ against the latent factors using ordinary least squares (or ridge) regression,

$$\mathbf{Y} \approx \mathbf{TQ}^T, \quad \mathbf{Q} = \mathbf{Y}^T\mathbf{T}(\mathbf{T}^T\mathbf{T})^{-1}.$$

d. Obtain the matrix $\mathbf{B}$ of regression coefficients:

$$\mathbf{Y} \approx \mathbf{XB}, \quad \mathbf{B} = \mathbf{WQ}^T = \mathbf{W}(\mathbf{T}^T\mathbf{T})^{-1}\mathbf{T}^T\mathbf{Y}.$$

We split genes into test and training sets for cross validation experiments. Training data including motif matrix $\mathbf{X}$ and gene expression matrix $\mathbf{Y}$ were used to learn matrix of regression coefficients $\mathbf{B}$. And we assessed predictive power of PLS on test data $\mathbf{Y}_{tst}$ and $\mathbf{X}_{tst}$ by normalized mean squared error (NMSE):

$$NMSE = \frac{E((\mathbf{X}_{tst}\mathbf{B} - \mathbf{Y}_{tst})^2)}{E((\mathbf{Y}_{tst} - \overline{\mathbf{Y}}_{tst})^2)} \tag{3.1}$$

where $E(\cdot)$ denotes the expected value and $\overline{\mathbf{Y}}_{tst} = E(\mathbf{Y}_{tst})$.

PLS not only provides a solution to the regression problem, but it also describes the covariance structure between $\mathbf{X}$ and $\mathbf{Y}$. It constructs K weight vectors $\mathbf{w}_i$ in the input space $\mathbb{R}^M$ and corresponding vectors $\mathbf{c}_i$ in the output space $\mathbb{R}^E$, where $\text{cov}(\mathbf{X}\mathbf{w}_i, \mathbf{Y}\mathbf{c}_i)$ is maximal. Intuitively, each weight vector $\mathbf{w}_i$ corresponds to a set of motifs (k-mers) that helps explain expression patterns in the direction $\mathbf{c}_i$. The k-mers with largest coefficients in $\mathbf{w}_i$ are the most important variables for predicting the projection of the expression patterns of genes onto $\mathbf{c}_i$.

SIMPLS algorithm

There are a number of variants of PLS, each of which defines and solves an optimization problem for constructing the weight matrix $\mathbf{W}$. We use the SIMPLS (Statistically Inspired Modification of PLS) algorithm Jong (1993), which optimizes an objective function defined on the matrix $\mathbf{Y}^T\mathbf{X}$. The latent factors $\mathbf{t}_i, i = 1, \ldots, K$ in T are sequentially built by estimating weight vectors $\mathbf{w}_i$ as follows:

For $i = 1, \ldots, K$:

a. Maximize the covariance between $\mathbf{t}_i$ and $\mathbf{Y}$:

$$\mathbf{w}_i = \text{argmax}_{\mathbf{w}} \text{cov}(\mathbf{Y}, \mathbf{t})^2 = \text{argmax}_{\mathbf{w}} \mathbf{w}^T\mathbf{X}^T\mathbf{Y}\mathbf{Y}^T\mathbf{X}\mathbf{w} \tag{3.2}$$

where $\mathbf{w}_i$ is a unit vector.

b. Impose orthogonality constraints $\mathbf{t}_i^T \mathbf{t}_j = \mathbf{w}_i^T \mathbf{X}^T \mathbf{X} \mathbf{w}_j = 0$ for all $j = i + 1, \ldots, K$, by deflating $\mathbf{Y}^T \mathbf{X}$:

$$\mathbf{Y}^T \mathbf{X} = \mathbf{Y}^T \mathbf{X} - \mathbf{v}_i (\mathbf{v}_i^T \mathbf{Y}^T \mathbf{X}) \tag{3.3}$$

where (i) If $i = 1$, $\mathbf{v}_i = \mathrm{norm}(\mathbf{X}^T \mathbf{t}_i)$.

(ii) If $i > 1$, $\mathbf{v}_i = \mathrm{norm}(\mathbf{X}^T \mathbf{t}_i - \mathbf{V}(\mathbf{V}^T \mathbf{X}^T \mathbf{t}_i)) \quad \mathbf{V} = [\mathbf{v}_1, ..., \mathbf{v}_{i-1}]$.

Regularized partial least squares regression

We now modify the PLS algorithm with the dual goals of (1) making the solution more interpretable and (2) regularizing the optimization problem, to reduce over-fitting. We impose two constraints to achieve these goals. First, we use a lasso (L^1) constraint Tibshirani (1996) to promote sparsity in the weight vectors $\mathbf{w}_i$, that is, drive the weights for many k-mers to zero. Sparsity is clearly attractive since fewer k-mers contribute to the solution, making it easier to identify the most important motifs. The lasso constraint over coordinates w^p of weight vector $\mathbf{w}$ takes the form:

$$||\mathbf{w}||_1 = \sum_{p=1}^{M} |w^p| \leq b_1 \tag{3.4}$$

For the second constraint, we want sequence-similar k-mers to have similar coefficients in the weight vectors, so that a group of similar k-mers are more likely to act as a single motif pattern in the regression problem. We define a graph structure on the k-mers where we place an edge $p \sim q$ if the Hamming distance between the pair of k-mers p and q is less than threshold σ. Since k-mers represent potential binding sites in double-stranded DNA, here we take the distance between two k-mers p and q to be the minimum of the Hamming distances $d(p, q)$ and $d(p, \mathrm{rc}(q))$, where $\mathrm{rc}(q)$ is the reverse complement of q. We then impose a smoothness constraint in the form of the graph Laplacian Weinberger et al. (2007), as described below.

The Laplacian matrix $\mathbf{L} = (\mathbf{L}^{pq})$ for an unweighted graph is defined as

$$\mathbf{L}^{pq} = \begin{cases} \deg(q) & \text{if } p = q, \\ -1 & \text{if } p \text{ is adjacent to } q, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

where $\deg(q)$ denotes the degree of k-mer q, the number of edges that connect k-mer q with other k-mers. If we write $\mathbf{w} = (w^p) \in \mathbb{R}^M$ as a column vector and view it as a function on the graph – i.e. a function that assigns a weight w^p to each vertex p – then we can use the graph Laplacian to compute a quadratic form on $\mathbf{w}$ that satisfies the relationship Chung (1997):

$$\mathbf{w}^T\mathbf{L}\mathbf{w} = \sum_{p\sim q} |w^p - w^q|^2. \tag{3.6}$$

Equation (3.6) shows that this quadratic form measures the *smoothness* of $\mathbf{w}$ with respect to the graph: the quadratic form is small when the function's values vary smoothly over adjacent nodes, so that the weights for sequence-similar k-mers are close in value. Therefore, the second constraint that we impose is precisely on the size of the quadratic form, enforcing smoothness on the weight vector $\mathbf{w}$:

$$\mathbf{w}^T\mathbf{L}\mathbf{w} \leq b_2. \tag{3.7}$$

A pseudocode description of the graph-regularized PLS algorithm is given in Figure 5.1.

Filtering k-mer features

k-mer features with very sparse genome-wide counts are unlikely to improve the loss function – since they only only in a handful of promoters – and can contribute to overfitting. In order to eliminate k-mers with infrequent counts prior to training, we filter the k-mer feature set based on expect counts on background sequences.

```
INPUT:
  $\mathbf{X}$ ($G \times M$, column normalized): motif matrix
  $\mathbf{Y}$ ($G \times E$, column normalized): expression matrix
  $\mathbf{S} = \mathbf{Y}^T\mathbf{X}$: cross-product
  K: number of latent factors

Algorithm:
  Loop over latent factors: For $i = 1, \ldots, K$
    (1) Learn weight vectors and latent factors:
      $\mathbf{w} = \mathrm{argmax}_{\mathbf{w}}(\mathbf{w}^T\mathbf{S}^T\mathbf{S}\mathbf{w})$, subject to
        (i) $\mathbf{w}^T\mathbf{w} = 1$
        (ii) $\sum_{p=1}^{M} |w^p| \leq b_1$
        (iii) $\sum_{p\sim q} |w^p - w^q|^2 \leq b_2$
      Compute latent factor: $\mathbf{t} = \mathbf{X}\mathbf{w}$
      Normalize latent factor: $\mathbf{t} = \mathbf{t}/\sqrt{\mathbf{t}^T\mathbf{t}}$
      Rescale weight vector: $\mathbf{w} = \mathbf{w}/\sqrt{\mathbf{t}^T\mathbf{t}}$
      $\mathbf{c} = \mathbf{Y}^T\mathbf{t}$
      $\mathbf{u} = \mathbf{Y}\mathbf{c}$
    (2) Deflate $\mathbf{S}$:
      $\mathbf{v} = \mathbf{X}^T\mathbf{t}$
      if $i > 1$ then
        $\mathbf{v} = \mathbf{v} - \mathbf{V}(\mathbf{V}^T\mathbf{v})$
      $\mathbf{v} = \mathbf{v}/\sqrt{\mathbf{v}^T\mathbf{v}}$
      $\mathbf{S} = \mathbf{S} - \mathbf{v}(\mathbf{v}^T\mathbf{S})$
      Store $\mathbf{w}$, $\mathbf{t}$, $\mathbf{c}$, $\mathbf{u}$ and $\mathbf{v}$ into column $i$ of $\mathbf{W}$, $\mathbf{T}$, $\mathbf{C}$, $\mathbf{U}$ and $\mathbf{V}$, respectively

OUTPUT:
  OLS regression from $\mathbf{T}$ to $\mathbf{Y}$:
    $\mathbf{Y}_{pred} = \mathbf{T}\mathbf{C}^T \quad \mathbf{C}^T = \mathbf{T}^T\mathbf{Y}$
    Regression matrix $\mathbf{B} = \mathbf{W}\mathbf{C}^T = \mathbf{W}\mathbf{T}^T\mathbf{Y}$
    $\mathbf{Y}_{pred} = \mathbf{X}\mathbf{B} = \mathbf{X}\mathbf{W}\mathbf{C}^T = \mathbf{T}\mathbf{C}^T = \mathbf{T}\mathbf{T}^T\mathbf{Y}$
```

Figure 3.1: **Pseudocode for graph-regularized PLS.** A pseudocode description of the iterative PLS procedure, enforcing sparsity and Laplacian constraints on motif weight vectors.

We denote the frequency of occurrence of motif m in background sequences p. Given the rate p, we evaluate the binomial probability of observing L occurrences of motif m out of total N k-mers of the same length in the promoter sequences. We report the probability as a Z-score defined as $Z_m = \frac{L-Np}{\sqrt{Np(1-p)}}$ Eskin et al. (2002), which measures the number of standard deviations away from what is expected by chance when the null model is assumed to be binomial. Motifs with high Z-score conservation are seen as frequently occurring.

We construct the background sequences by shuffling exon sequences 100 times and rank k-mers by the Z-score: $Z_m = \frac{L-Np}{\sqrt{Np(1-p)}}$, where L is the number of the k-mer in all promoter sequences, N is the length of all shuffle exon sequences, and $p = \frac{L_b}{N}$ is number of the k-mer in all shuffle exon sequences divide by N. (Note that shuffle intergenic sequences could also be use to generate the random model.) We keep the top 3000 k-mers and build the motif matrix containing counts of k-mers in promoter sequences. We find that this filtering step significantly improve cross-validation performance.

Hierarchical sequence agglomeration

For each latent factor **t**, we rank k-mers by their components in the corresponding weight vector **w** and perform motif analysis on the top 50 k-mers. Those k-mers are first displayed in the form of a motif graph via Cytoscape Shannon et al. (2003), in which an edge between two k-mer nodes indicates similarity. We use the MCODE Cytoscape Plugin Bader and Hogue (2003) to find k-mer clusters (highly interconnected sets of sequence-similar k-mers) in the graph. Each k-mer cluster represents a motif pattern consisting of slightly different k-mers.

Finally we perform a hierarchical sequence agglomeration algorithm that iteratively merges motifs that are most similar into a single motif, represented by a probabilistic model called position-specific scoring matrices (PSSMs) for k-mer clusters. For a given sequence length n_p, a position-specific scoring matrix (PSSM) is defined as a probability distribution $p(x_1, x_2, ..., x_{n_p})$ over sequences $x_1 x_2 ... x_{n_p}$, where

$x_i \in \{A, C, G, T\}$. The emission probabilities are assumed to be independent at every position such that $p(x_1, ..., x_{n_p}) = \prod_{i=1}^{n} p_i(x_i)$, where $\sum_{x \in \{A,C,G,T\}} p_i(x) = 1$ for all i. The PSSM n is thus defined by a set of probabilities $\{p_i(x)\}_{i \in 1,...,n_p, x \in \{A,C,G,T\}}$. Every PSSM is associated with a log-odds score S_{n_p} for a given sequence $x_1, ..., x_{n_p}$:

$$S_{n_p}(x_1, ..., x_{n_p}) = \sum_{i=1}^{n} log \frac{p_i(x_i)}{p^{bg}(x_i)} \tag{3.8}$$

where p^{bg} gives the background nucleotide probabilities for smoothing. Within each k-mer cluster, each k-mer is treated as a seed PSSM (using background nucleotide probabilities for smoothing), and then the algorithm iteratively merges similar PSSMs until a single PSSM is learned as the binding site model.

When comparing two PSSMs p and q of equal length $n_p = n_q$, we use the Kullback-Leibleer divergence D_{KL}. Given that the position-specific probabilities are independent, one can easily show

$$\begin{aligned} D_{KL}(p||q) &= \sum_{x_1,..,x_{n_p}} p(x_1, ..., x_{n_p}) log \frac{p(x_1, ..., x_{n_p})}{q(x_1, ..., x_{n_p})} \\ &= \sum_{i=1}^{n_p} \sum_{x_i \in \{A,C,G,T\}} p_i(x_i) log \frac{p_i(x_i)}{q_i(x_i)} \\ &= \sum_{i=1}^{n_p} D_{KL}(p_i||q_i) \end{aligned} \tag{3.9}$$

where the summation goes over all possible sequencs $x_1, ..., x_{n_p}$ with every $x_i \in \{A, C, G, T\}$. When merging two PSSMs, the starting position of the first PSSM does not necessarily have to coincide with the starting position of the second PSSM. Instead, we allow offsets between their starting positions and pad either the left or right ends with the background distribution. For PSSMs with arbitrary lengths n_p and n_q (not always equal) and with offsets $l \in \{-n_q, ..., n_p\}$ in starting positions, we define

$$D_{KL,l}(p||q) = \sum_{i=1}^{max(n_p-l,n_q)+max(0,-l)} D_{KL}(p_i + min(l,0)||q_i + min(0,-l)) \quad (3.10)$$

where we pad PSSMs for positions $i \leq 0$ and $i > n_p$ with background probabilities to let them have equal lengths. Finally we can define a distance measure $d(p,q)$ as the minimum over all possible position offsets of the JS entropy.

$$xd(p,q) = \min_{\text{offsets}} [h_p D_{KL}(p|h_p p + h_q q) + h_q D_{KL}(q|h_p p + h_q q)], \quad (3.11)$$

The relative weights of the two PSSMs, h_p and h_q, are here defined as $h_{p,q} = N_{p,q}/(N_p + N_q)$, where N_p, N_q are the numbers of target genes for the given PSSM. The initial PSSMs are k-mers and the number of target genes are the number of promoter sequences with the k-mer occurrence. The number of target genes for the newly merged PSSM will be the number of target genes combined for the two old PSSMs.

Assigning genes to latent factors

To extract biological information from the algorithm output, we analyze latent factors for potential gene groups and corresponding biological functions. To do that, we assign each gene g to the gene group associated with a factor i based on $\mathbf{TU}$ values. Here, the matrix $\mathbf{T}$ (respectively, $\mathbf{U}$) is formed by placing vectors $\mathbf{t}_i$ (respectively, $\mathbf{u}_i$) for latent factors $i = 1 \ldots 5$ as column vectors (Figure 3.3). The value $\mathbf{T}_{gi}$ indicates how well $\mathbf{w}_i$ captures the k-mer profile of gene g, and the value $\mathbf{U}_{gi}$ measures the similarity between $\mathbf{c}_i$ and expression profile of gene g. In contrast to traditional clustering, which only relies on gene expression to group genes, we integrate both sequence and gene expression information in learning potentially functional gene sets. For each gene g, we computed $\mathbf{T}_{gj}\mathbf{U}_{gj}$ across all factors and chose factor i with the maximum value:

$$i = \text{argmax}_j \mathbf{T}_{gj}\mathbf{U}_{gj}, \qquad j = 1 \ldots 5 \qquad \text{subject to } \mathbf{T}_{gi}, \mathbf{U}_{gi} > 0.$$

Since we suspect that only large $\mathbf{T}_{gi}\mathbf{U}_{gi}$ values indicate strong association of a gene g with factor i, we assign gene g to factor i only when $\mathbf{T}_{gi}\mathbf{U}_{gi}$ was in the top 20% of all $\mathbf{TU}$ values. Although we use $K = 4$ latent factors in our model, here we compute the representation with five factors, reasoning that if a gene is assigned to the 5th factor, it should not be included in our main analysis.

Conservation of motifs

To look for evidence of the functional roles of highly weighted motifs in PLS regression, we considered conservation patterns of these sequences. *Caenorhabditis briggsae* is closely related to *C. elegans* and is frequently used in comparative genomics studies in worm. One expects that motifs responsible for a biological function that is shared by the two species, such as oogenesis, would be under evolutionary pressure and therefore conserved in the promoter regions of orthologous genes contributing to this function. We studied the genome-wide conservation of all k-mers, based on their frequent conservation between the two species. A conserved occurrence of a k-mer m is an instance of the k-mer for which an exact match to the k-mer is present in both species. We first defined the conservation rate p of a k-mer as the number of occurrences of k-mer which are are conserved across the two species divided by total number of occurrences of k-mer in *Caenorhabditis briggsae*. We then calculated a Motif Conservation Score (MCS) based on the conservation rate of each k-mer in the promoter regions. To evaluate the Motif Conservation Score (MCS) of a k-mer m of given length, we compared its conservation rate p to expected rate p_o for similar random k-mers of the same length. Assuming the underlying null model is binomial, we reported the MCS as a Zscore ($MCS = \frac{L-Np_0}{\sqrt{Np_0(1-p_0)}}$) measuring the number of standard deviations of conserved instances away from what is expected by chance. Motifs that have high motif conservation scores, are both highly conserved and frequently occurring. To estimate the expected conservation rate p_0 for a k-mer of given length, we obtain the averaged conservation rate of 500 random motifs of the same length. To take into account the nucleotide compositional bias in the promoter regions, we generate these random motifs by sampling from the background distribution of nucleotides in promoters of all genes.

3.4 Results

3.4.1 Statistical Validation

3.4.1.1 Regularized PLS predicts held-out gene expression

We performed 10-fold cross-validation experiments, randomly splitting genes into test and training sets with 10% of the data assigned to test data. Figure 3.2 illustrates the normalized mean squared error on the cross-validation test sets versus number of latent factors for both standard and graph-regularized PLS. Here, the mean squared error obtained with zero latent factors (i.e. the variance of the test data) is normalized to 1, so that cross-validation errors below 1 indicate that the model is explaining part of the variance of the held-out data. Figure 3.2 shows the average mean squared error across the cross-validation folds with the standard deviation over folds indicated with error bars.

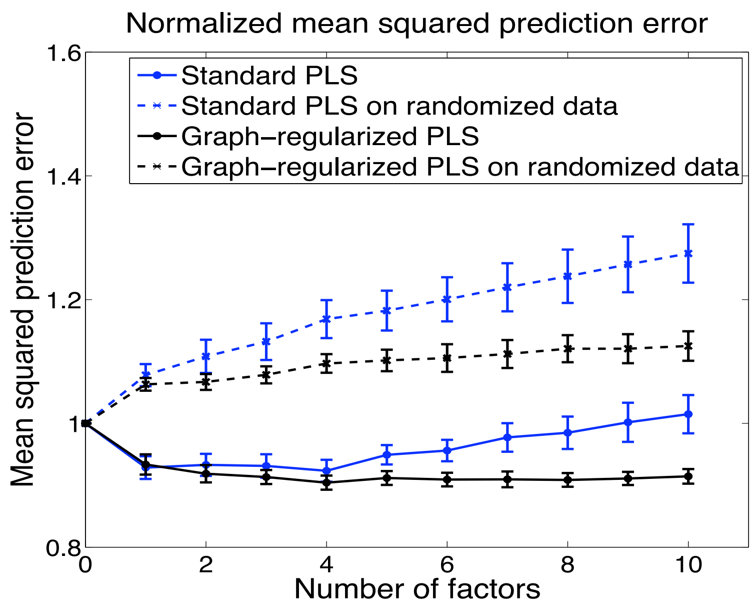


Figure 3.2: **Normalized mean squared error on cross-validation test data.** Normalized mean squared error versus number of latent factors for standard PLS and graph-regularized PLS on real and randomized data. For the real cross-validation data, standard PLS overfits after the 4th factor; graph-regularized PLS is more resistant to overfitting than standard PLS. As expected, when trained and tested on randomized data, both standard and graph-regularized PLS overfit with the very first factor.

The minimal cross-validation error with standard PLS is obtained with four latent factors. Graph-regularized PLS appears to be more resistant to overfitting, with slightly lower cross-validation error at four latent factors and no substantial increase in error as the number of latent factors increases. Again, cross-validation error suggests that four latent factors should be used in the model. As a negative control, we randomly paired promoter sequences with expression profiles, so that we used real expression data and promoter sequences but lost the correspondence between sequence and expression, and we performed standard PLS and graph-regularized PLS . As can be seen from Figure 3.2, both standard PLS and graph-regularized PLS on randomized data overfit with the very first latent factor, indicating that the performance obtained on the real data is meaningful.

3.4.2 Biological Validation

3.4.2.1 Learning graph-mer motifs and corresponding expression trajectories

In order to learn the correspondence between (sets of) regulatory motifs in the promoter sequences of genes and gene expression trajectories over a time course, we posed a regression problem: using a training set of G genes, learn a linear mapping from the vector of counts of k-mer occurrences in a gene's promoter to the gene's time course expression profile. This model can then be used to predict expression from sequence on held-out genes, and k-mer features that are highly weighted in the model should represent important regulatory motifs. Here we have a very high-dimensional input space of motifs (k-mers) as well as a multivariate output space, both of which rule out use of ordinary least squares regression. Instead, our algorithm makes use of a partial least squares (PLS) regression strategy. PLS is a well-known statistical technique for fitting linear models when the input space is high dimensional Boulesteix and Strimmer (2007) and has both univariate and multivariate formulations.

Standard PLS represents the input data as a motif matrix $\mathbf{X}$ (dimension $G \times M$, where M is the number of k-mers), representing k-mer counts for each gene's promoter, and the gene expression matrix by $\mathbf{Y}$ (dimension $G \times E$, where E is the number of experiments), and then it performs two basic steps (see Methods for more details):

a. Construct K *weight* vectors $\mathbf{w}_1 \cdots \mathbf{w}_K$ in $\mathbb{R}^M$ and corresponding *latent* factors $\mathbf{t}_1 \cdots \mathbf{t}_K$ in $\mathbb{R}^G$, where the weight vectors are chosen so that the latent factors have maximal covariance with directions in $\mathbf{Y}$. The latent factors define a reduced dimensional representation of the promoter sequence data.

b. Regress $\mathbf{Y}$ against the latent factors using ordinary least squares (or ridge) regression. The latent factor dimensionality reduction followed by linear mapping to $\mathbf{Y}$ yields the final mapping from sequence to expression.

PLS algorithms typically work iteratively, so that each round i generates a new latent factor, and the number of rounds K is chosen by cross-validation to minimize the square loss function in the regression problem.

Here, we are most interested in what PLS tells us about the covariance structure between $\mathbf{X}$ and $\mathbf{Y}$ and how to interpret this information in terms of sequence motifs and expression patterns. In particular, along with K weight vectors $\mathbf{w}_i$ in the input motif space, PLS determines corresponding vectors $\mathbf{c}_i$ in the output expression space, defined so that $\mathrm{cov}(\mathbf{X}\mathbf{w}_i, \mathbf{Y}\mathbf{c}_i)$ is maximal (Figure 3.3). Intuitively, each weight vector $\mathbf{w}_i$ corresponds to a set of motifs (k-mers) that helps explain expression patterns in the direction $\mathbf{c}_i$. The components of the vector $\mathbf{w}_i$ that have large positive weights are the k-mers that most strongly predict the expression pattern $\mathbf{c}_i$.

To obtain a more interpretable model, we mathematically imposed two additional requirements on the PLS solution. First, we wanted the weight vectors $\mathbf{w}_i$ to be sparse, i.e. we wanted relatively few k-mers to have non-zero components, so that the algorithm produces a small number of hopefully functional motifs. Second, for each weight vector $\mathbf{w}_i$, we wanted sequence-similar k-mers to have similar weights, since such k-mers may represent variants of the same binding site and potentially should contribute in the same way to the linear model. We achieved the first goal by adding a lasso constraint to the PLS optimization problem (see Methods for more details). For the second goal, we defined a graph on the set of k-mers, joining two k-mers by an edge exactly when they are close in Hamming distance, and imposed a graph Laplacian constraint to obtain smoothness over the graph (see Methods for more details). Incorporating these constraints into a multivariate PLS approach yields a new algorithm that we call graph-regularized PLS.

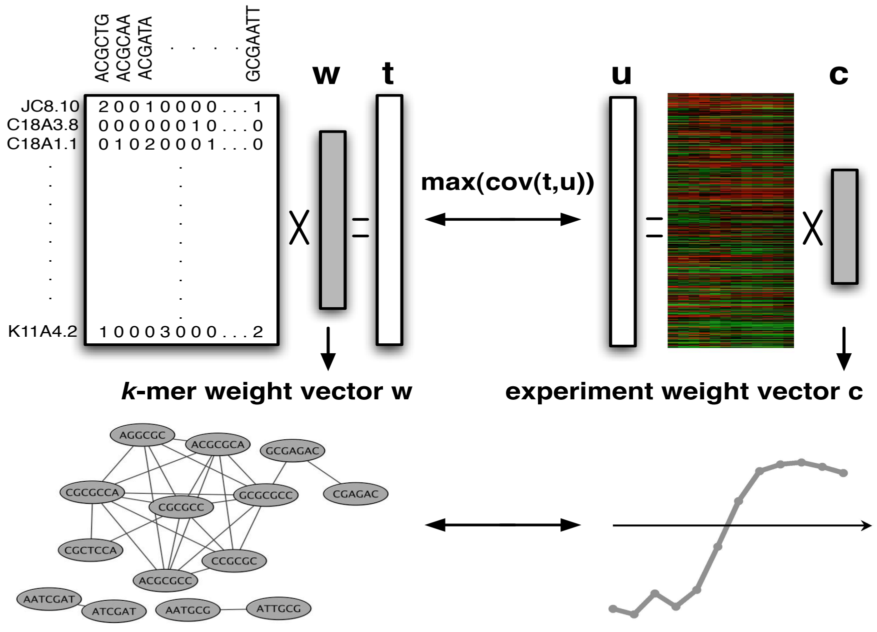


Figure 3.3: **Mapping between motif weight vectors and experiment weight vectors.** At each iteration i of the modified PLS algorithm, $i = 1 \ldots K$, weight vectors $\mathbf{w}_i$ and $\mathbf{c}_i$ are derived by finding latent factors $\mathbf{t}_i$ and $\mathbf{u}_i$ with maximal covariance. For clarity, subscripts i are omitted in the diagram and in the rest of the description. Each weight vector $\mathbf{w}$ is a vector in $\mathbb{R}^M$, where M is the number of k-mers used as input to the algorithm. Due to graph-regularization, each weight vector is sparse, i.e. most k-mers have weight 0, and smooth over a graph connecting sequence-similar k-mers, i.e. similar k-mers get assigned similar weights. Therefore, we can visualize the weight vector as a "graph-mer", a graph where nodes correspond to k-mers with high positive weights and edges connect sequence-similar k-mers (bottom left). At each iteration, the PLS procedure sets up a correspondence between the motif weight vector $\mathbf{w}$ and a weight vector over expression experiments represented by vector $\mathbf{c}$. In our setting, the series of expression experiments is a time course, and the vector $\mathbf{c}$ can be viewed as an expression pattern or trajectory (bottom right). Intuitively, we can think of the set of k-mers shown in the graph-mer as driving the expression pattern $\mathbf{c}$.

With these additional constraints, we can view the motif vectors $\mathbf{w}_i$ as "graph-mers" – weighted graphs over k-mers, where highly weighted dense clusters in the graphs correspond to important sequence-similar k-mer sets, or motifs. Figure 1 illustrates the mapping between motif weight vectors, interpreted as graph-mers, and corresponding expression patterns, arising from the latent factors found in graph-regularized PLS. Intuitively, we can think of each vector $\mathbf{c}_i$ as the expression pattern driven by the positively weighted k-mers in $\mathbf{w}_i$, that is, the common expression trajectory displayed by genes containing these motifs. This correspondence will be important for interpreting regulatory motifs in worm germline development below.

3.4.2.2 Latent factors map to germline-specific expression trajectories

By analyzing separate microarray expression data from germline mutants, the previous study also identified two gene sets consisting of sperm and oocyte genes Reinke et al. (2004), which we used in our analysis of the wild type developmental gene expression profiles. First, we estimated the prediction error on each gene set as shown in Figure 3.4. Clearly, the first and second latent factors account for the largest loss reduction for oocyte and sperm genes, respectively. To show that the first two factors dominate these two gene sets, we first examined the expression profiles of the two gene sets. In PLS, each weight vector $\mathbf{c}_i$ gives the weights over time points and can be interpreted as an expression pattern, and genes significantly influenced by the latent factor tend to follow this expression pattern. We plot the oocyte gene expression profiles together with $\mathbf{c}_1$ and sperm gene expression profiles with $\mathbf{c}_2$ in Figure 3.5(a) and 3.5(b). The gene expression profiles are strongly correlated with the corresponding weight vectors, indicating that the first two factors are able to retrieve the expression patterns of these two gene sets, respectively.

Furthermore, we used functional enrichment analysis to confirm that the genes identified statistically by these two factors are indeed enriched for oocyte or sperm genes, respectively. Given a gene set S and a real-valued ranking of all

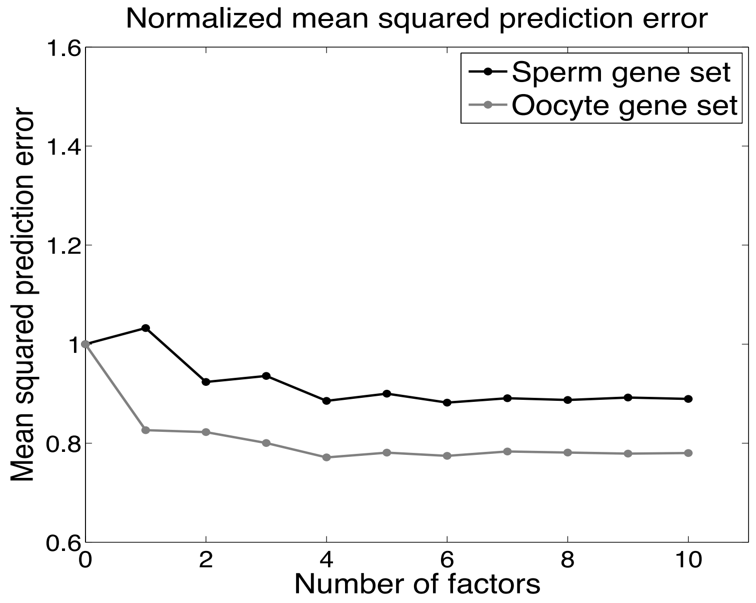
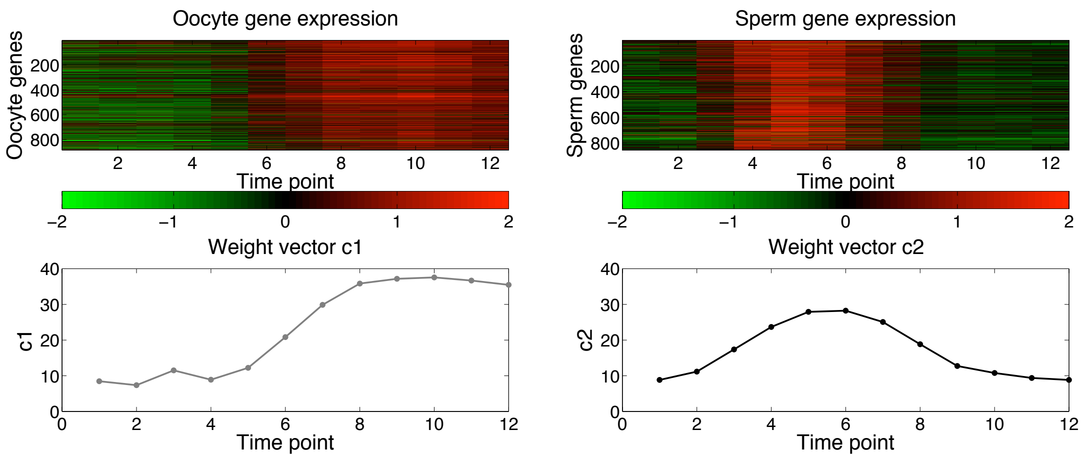


Figure 3.4: **Normalized mean squared error on cross-validation test data.** Normalized mean squared error versus number of latent factors for standard PLS and graph-regularized PLS on real and randomized data. The mean squared error obtained with zero latent factor is normalized to 1. Computed standard deviations of squared error across cross-validation sets are plotted as error bars. For the real cross-validation data, standard PLS overfits after the 4th factor; graph-regularized PLS is more resistant to overfitting than standard PLS. As expected, when trained and tested on randomized data, both standard and graph-regularized PLS overfit with the very first factor.

genes, we can use a procedure similar to gene set enrichment analysis (GSEA) Subramanian et al. (2005) to establish whether the empirical cumulative distribution of genes in S is significantly shifted up or down compared to the set of genes not in S. Here, we use sperm and oocyte genes as gene sets and use either correlation with $\mathbf{c}_i$ or number of k-mer hits (for the top 50 graph-mer k-mers in $\mathbf{w}_i$) to produce the ranking. Figure 3.6(a,b) plots the empirical CDF of the correlation between gene expression and $\mathbf{c}_i$, $(i = 1, 2)$, showing that oocyte and sperm gene sets are enriched toward the top of the gene expression correlation. Similarly, Figure 3.6(c,d) plots the empirical CDF for k-mer hits, showing that oocyte and sperm gene sets are enriched in the corresponding k-mer hits. These results indicate that graph-regularized PLS can be used in conjunction with gene set analysis to identify functional categories that are supported both by shared motif information and expression trajectories.

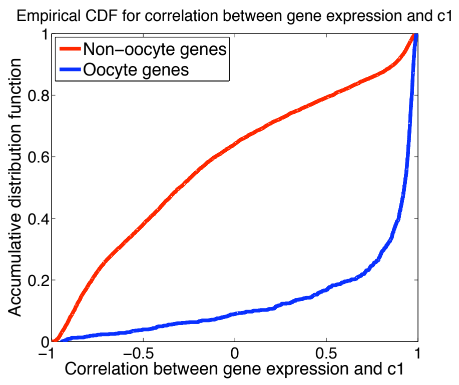


(a) (b)

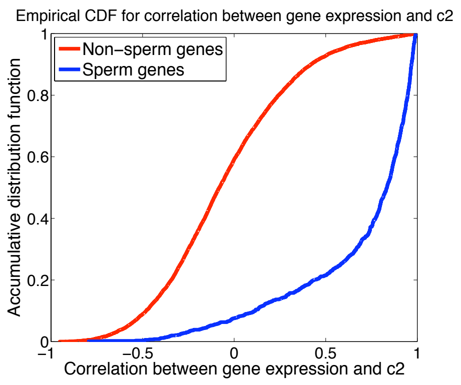
Figure 3.5: **Correlation of germ cell expression patterns and PLS expression weight vectors.** Oocyte and sperm gene expression patterns are strongly correlated with c_1 and c_2, respectively. (a) Oocyte gene expression versus c_1. (b) Sperm gene expression versus c_2.

3.4.2.3 Interpretation of motif weight vectors

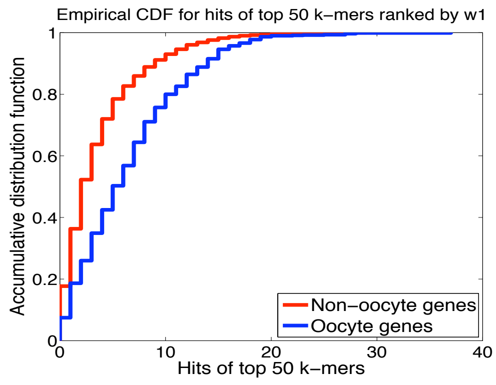
In PLS, each weight vector $\mathbf{w}_i$ corresponds to a set of motifs (k-mers) that help to explain expression patterns in the direction $\mathbf{c}_i$. The k-mers with largest coefficients in $\mathbf{w}_i$ are the most important variables for predicting the projection of the expression patterns of genes onto $\mathbf{c}_i$. To identify motifs relevant for sperm and oocyte gene sets, we selected the top 50 k-mers ranked by $\mathbf{w}$ and examined the k-mer graphs corresponding to the first two latent factors. Clusters in the graph that are identified by MCODE Bader and Hogue (2003) represent motif patterns and hierarchical sequence clustering is performed to generate corresponding PSSMs. Figures 3.7



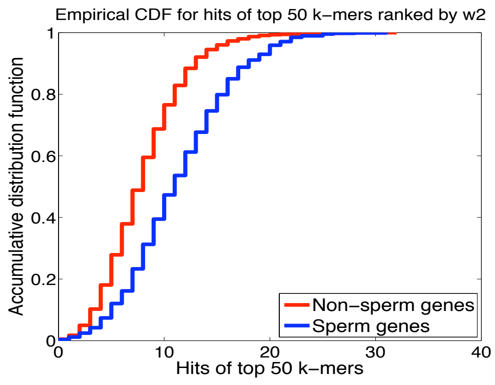
(a)



(b)



(c)



(d)

Figure 3.6: **Correspondence between first and second latent factors and sperm and oocyte genes.** (a,b) The set of all genes is split into oocyte and non-oocyte genes, or sperm and non-sperm genes, and the empirical cumulative distribution of correlation with $\mathbf{c}_i$, $i = 1, 2$ is plotted. Oocyte and sperm genes are enriched towards the top of the gene expression correlation distribution. (c,d) The set of all genes is split into oocyte and non-oocyte genes, or sperm and non-sperm genes, and the corresponding empirical cumulative distributions of hits of top 50 k-mers in $\mathbf{w}_i$, $i = 1, 2$ are plotted. Oocyte and sperm genes are enriched in k-mer hits corresponding to the 1st and 2nd weight vectors.

and 3.8 show the graph-mer representation of the top 50 k-mers, motif patterns and PSSMs for the first two factors.

From the second factor, we successfully found the ELT-1 ('erythrocyte-like transcription factor') motif GATAA and bHLH ('basic helix-loop-helix') motif

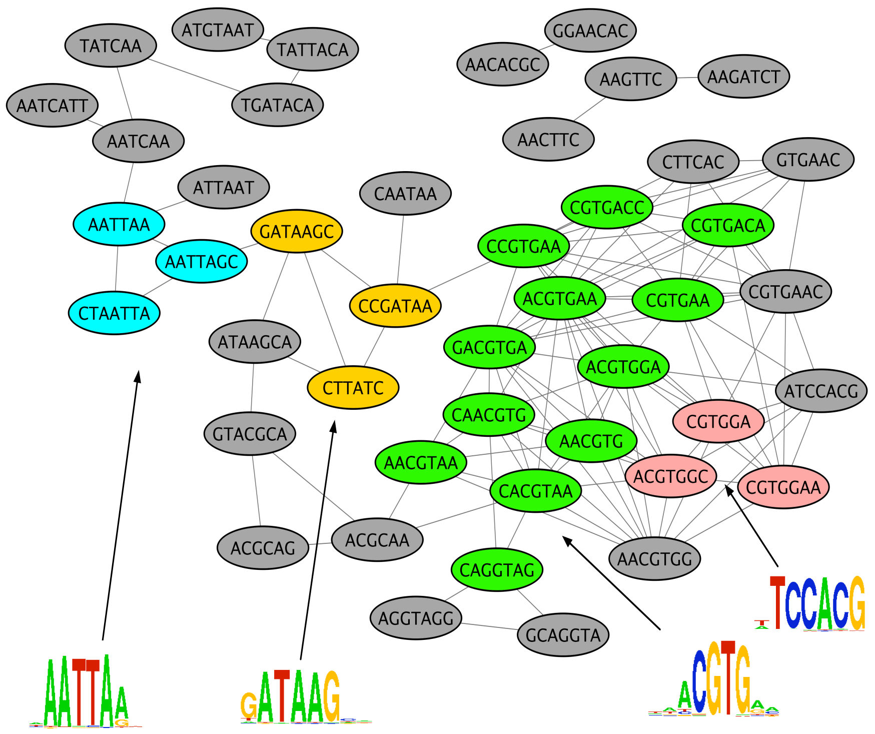


Figure 3.7: **Sperm motifs determined by graph-mer analysis and positional bias of motif ACGTG.** Sperm motifs extracted from graph-mer output. The graph-mer consisting of the top 50 k-mers ranked by w_2. Graph motif patterns identified in the form of k-mer clusters using the MCODE plug-in Bader and Hogue (2003) in Cytoscape are shown in different colors, with each subgraph summarized by a PSSM generated through hierarchical sequence agglomeration of the corresponding k-mers. Both the ELT-1 motif GATAA and the bHLH motif ACGTG are found in this way.

ACGTG, as shown in Figure 3.7. The ELT-1 protein is a transcriptional activator that can recognize the GATA motif, is highly expressed in the germ line, and has as potential targets a number of genes encoding major sperm proteins Shim (1999). The bHLH proteins act through E-box elements with consensus CANNTG; the canonical E-box is CACGTG. bHLH proteins have been found to act at the E-box and influence hormone-induced promoter activation in mammalian Sertoli cells, which are required to maintain the process of spermatogenesis J. and K. (1999);

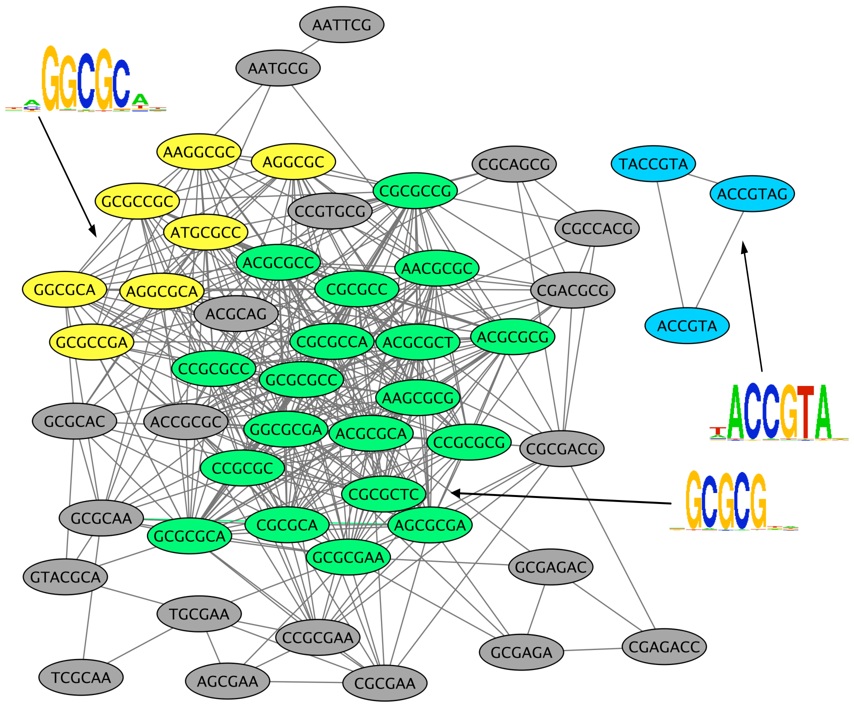
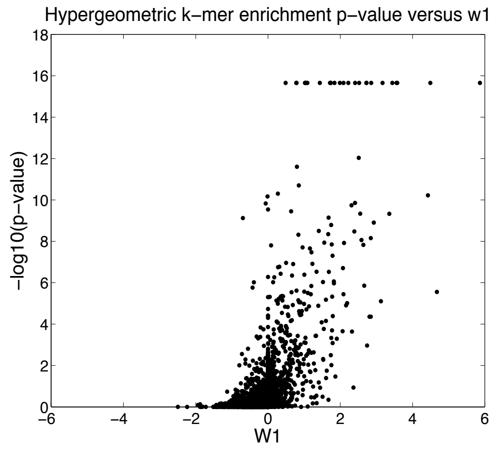


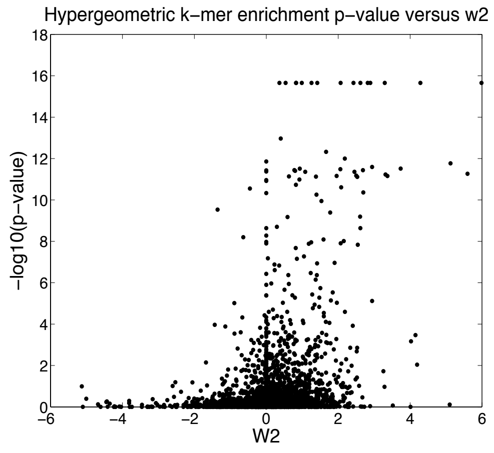
Figure 3.8: **Oocyte motifs determined by graph-mer analysis and conservation of graph-mer derived oocyte and sperm motifs.** Top 50 k-mers ranked by the weight vector $\mathbf{w}_1$, depicted as a graph-mer, which are associated by the PLS procedure to the expression pattern of oocyte genes. Graph motif patterns were identified in the form of k-mer clusters using the MCODE plug-in in Cytoscape. PSSMs generated through hierarchical sequence agglomeration of the corresponding k-mer sets are indicated, revealing several CG-rich motifs.

however, this motif has not previously been associated with spermatogenesis in *C. elegans*.

For the first latent factor, the top ranked motifs are CG-rich sequences as shown in Figure 3.8. To evaluate the statistical significance of those k-mers for oocyte genes, we studied the enrichment of all k-mers in oocyte genes. For each k-mer, a hypergeometric distribution p-value was estimated based on the counts of oocyte genes and all genes having the k-mer's presence. Figure 3.9 plots the hypergeometric distribution $-log_{10}(p\text{-value})$ representing k-mer enrichment in oocyte genes versus the k-mer's $\mathbf{w}_1$ value. We found a moderate correlation (Pearson coefficient = 0.65) between the two variables, and in particular the k-mers highly ranked by $\mathbf{w}_1$ had p-values between 10^{-16} and 10^{-4}. This type of k-mer enrichment further validates the relevance of inferred k-mers from the first factor to oocyte genes. Similarly, we studied the enrichment of all k-mers in sperm genes and plotted the the hypergeometric distribution $-log_{10}(p\text{-value})$ representing k-mer enrichment in sperm genes versus the k-mer's $\mathbf{w}_2$ value (Figure 3.9(b)). There was some positive correlation between $-log_{10}(p\text{-value})$ and $\mathbf{w}_2$ (Pearson coefficient = 0.35), but it was weaker than that of oocyte genes.



(a)



(b)

Figure 3.9: **Correlation of weights with significance of enrichment in oocyte and sperm genes for the k-mers from 1st and 2nd graph-mer respectively.** We plot the weights of k-mers in the first motif weight vector versus the $-log_{10}(p\text{-value})$ for the enrichment of these k-mers in oocyte and sperm genes, as computed by the hypergeometric distribution. (a) For oocyte genes, $-log_{10}(p\text{-value})$ is moderately correlated with $\mathbf{w}_1$ (Pearson coefficient = 0.65), and k-mers highly ranked by $\mathbf{w}_1$ had p-values between 10^{-16} and 10^{-4}. This enrichment supports the functional relevance of PLS-derived k-mers from the first factor in oocyte genes. (b) For sperm genes, $-log_{10}(p\text{-value})$ is somewhat correlated with $\mathbf{w}_2$ (Pearson coefficient = 0.35), though the correlation is weaker than that of oocyte genes.

3.4.2.4 Positional bias and conservation of motifs

Since functional motifs sometimes exhibit a spatial bias in the promoter region – for example, overrepresentation close to the transcription start site (TSS) – we performed positional analysis of top ranked motifs by examining their distance to the TSS in sperm genes versus non-sperm genes. We observed that the sequence element ACGTG displayed strong positional bias towards the TSS of sperm genes. Figure 3.10 plots the distribution of distance of ACGTG to TSS in sperm genes versus non-sperm genes, showing that ACGTG is found far more frequently within 200bp upstream of the TSS of sperm genes but displays a fairly uniform distribution relative to TSS in non-sperm genes. This result indicates that motif ACGTG was significantly overrepresented immediately upstream of sperm genes, giving us additional confidence in the motif's contribution to sperm gene expression.

To look for evidence of the functional roles of CG-rich and other highly weighted motifs, we considered conservation patterns of these sequences. *Caenorhabditis briggsae* is closely related to *C. elegans* and is frequently used in comparative genomics studies in worm. One expects that motifs responsible for a biological function that is shared by the two species, such as oogenesis, would be under evolutionary pressure and therefore conserved in the promoter regions of orthologous genes contributing to this function. We studied the conservation of all k-mers between the two species and found that highly ranked k-mers, where rankings are induced by the 1st and 2nd factor, tended to be more conserved in the oocyte genes and sperm genes, respectively. Specifically, we computed the motif conservation score (MCS) Waterston et al. (2002) of each k-mer by comparing its conservation rate p to its expected rate p_0, estimated using 500 random k-mers of the same length. A conserved occurrence of a k-mer is an instance of the k-mer in the *C. elegans* genome, for which it is also present in the *C. briggsae* ortholog. We reported MCS as a Z-score ($MCS = \frac{L-Np_0}{\sqrt{Np_0(1-p_0)}}$) measuring the significance of observing

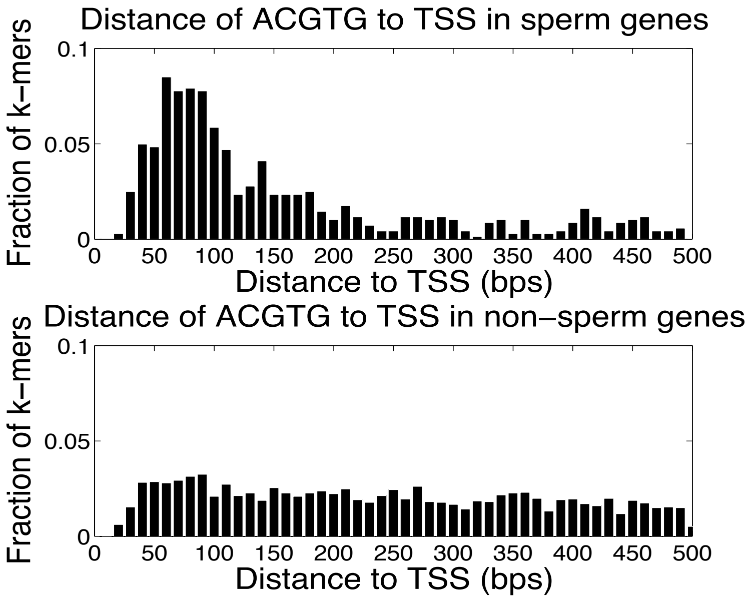
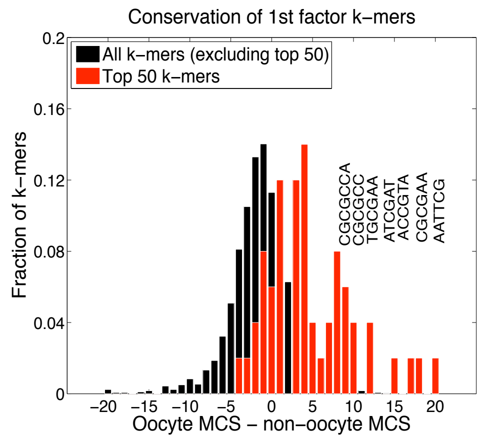
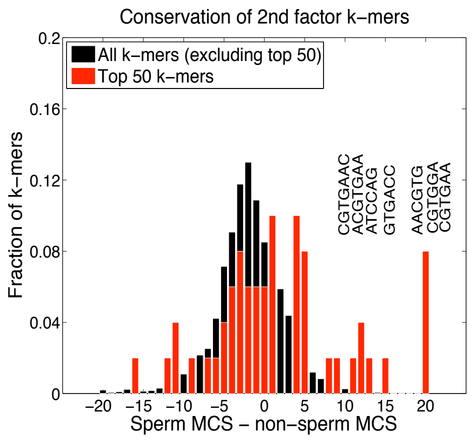


Figure 3.10: **Sperm motifs determined by graph-mer analysis and positional bias of motif ACGTG.** Distribution of distance of motif ACGTG to TSS (measured in base pairs) in sperm genes versus non-sperm genes. Motif ACGTG occurs more frequently within 200bp upstream of the TSS in sperm genes relative to non-sperm genes, giving us more confidence in its contribution to sperm gene expression.

L conserved occurrences out of total N occurrences. To assess the significance of inferred k-mers for oocyte and sperm gene sets, we focused on motif conservation in sperm and oocyte genes relative to non-sperm and non-oocyte genes. To do this, we computed the MCS of each k-mer in both oocyte genes and non-oocyte genes, and we plotted the distribution of the difference of these two MCS scores for top 50 ranked k-mers in the w_1 versus remaining k-mers, as shown in Figure 3.11(a); Figure 3.11(b) shows the difference of the MCS scores for sperm genes and non-sperm genes for the top 50 ranked k-mers in w_2 versus the remaining k-mers. For both oocyte and sperm gene sets, the score distribution for the top 50 k-mers has a heavy right tail relative to other k-mers, showing that the top k-mers have higher oocyte- and sperm-specific conservation. To confirm the significance of this observation, we performed a one-sided Kolmogorov-Smirnov (KS) test and found that the rightward shift was highly significant in both cases ($p < 3.0e$-13 and $p <$ 1.9e-5 for oocyte and sperm k-mers, respectively). The k-mers that are most significantly conserved in oocyte and sperm genes, relative to non-oocyte and non-sperm genes, are also annotated in Figure 3.11; these include the ACGTG motif for sperm genes and CG-rich k-mers for oocyte genes.



(a)



(b)

Figure 3.11: **Conservation of graph-mer derived oocyte and sperm motifs.** (a) Analysis of oocyte k-mer conservation using the motif conservation score (MCS). The plot shows the distribution of (oocyte MCS−non-oocyte MCS) for top 50 k-mers versus remaining k-mers in w_1. The score distribution for the top 50 k-mers has a heavy right tail, showing that as a distribution, the top 50 k-mers have higher oocyte-specific conservation scores as compared to other k-mers ($p <$ 3.0e-13 by a one-sided KS statistic). Significantly conserved k-mers are annotated, including CG-rich k-mers for oocyte genes. (b) Distribution of (sperm MCS−non-sperm MCS) for top 50 k-mers versus remaining k-mers in $\mathbf{w}_2$. The score distribution for the top 50 k-mers has a heavy right tail, showing that the top 50 k-mers have higher distribution of sperm-spefic conservation scores than other k-mers ($p <$ 1.9e-5, one-sided KS statistic). Significantly conserved k-mers are annotated, including ACGTG motif for sperm genes.

Targets of CG-rich motifs are expressed in the germline

Relatively little is known about transcriptional regulation of oocyte genes. To gain additional evidence supporting a functional role for learned motifs, we examined the *in situ* expression patterns of genes enriched with those motifs. We searched for a subset of EST (expressed sequence tag) clones known as YK clones of each gene in WormBase (http://www.wormbase.org) and looked at *in situ* expression patterns at the L4-adult stage associated with each YK clone in the Nematode Expression Pattern Database (NEXTDB http://nematode.lab.nig.ac.jp/db2/index.php).

The *in situ* analysis provides direct evidence about where the genes are expressed, and we expect that genes highly ranked by motif hits are more likely to be germline expressed. To obtain a ranked gene list for each of the three motifs in Figure 5A, we first defined the gene group associated with the first factor based on **TU** values (see Methods). For each motif, we ranked genes within the gene group by counts of k-mers of that motif and came up with a list consisting of top $\sim$80 genes. Table 1 summarizes the *in situ* expression patterns of genes associated with motif 1 (GGCGC), motif 2 (GCGCG) and motif 3 (ACCGTA). We split each gene list into two groups, those already known to be oocyte genes, and genes with high motif scores not already defined as oocyte genes. For each group, Table 1 shows number of genes examined; the number of genes with an *in situ* pattern; and percentage of genes expressed in germline tissues only, in both germline and somatic tissues, and somatic tissues only.

Over all three motifs, 7% of the genes have detectable *in situ* staining. Of those, an average of 78% stain only in the germ line, and with more than 80% of genes previously identified as oocyte genes staining in the germ line.

More than 70% of genes that had not previously been identified as oocyte genes (based on mutant expression profiling) were also dominantly expressed in germline tissues rather than somatic tissues. In the study that defined the oocyte

Motif	Previously identified as oocyte genes	# genes	# genes with *in situ* pattern	% Germline only	% Germline & somatic	% Somatic only
Motif 1	yes	29	28	71%	7%	5%
(GGCGC)	no	52	37	73%	8%	13%
Motif 2	yes	31	25	80%	4%	4%
(GCGCG)	no	55	43	74%	14%	5%
Motif 3	yes	26	16	94%	0%	0%
(ACCGTA)	no	62	38	76%	10%	0%

Table 3.1: ***In situ* analysis of genes enriched with CG-rich motifs.** For each graph-mer derived motif, we identified the set of genes associated to the motif based on latent factor analysis (see Methods). Each gene list was further split into two sets: genes that had been previously identified as oocyte genes based on mutant expression data and those not identified as oocyte genes by this previous analysis. The table shows the number of genes associated to the motif; the number of genes having an *in situ* pattern in the NEXTDB database; and genes expressed in germline tissues only, in both germline and somatic tissues, and somatic tissues only as a percentage of genes with an *in situ* pattern. The results show that even among genes not previously identified as oocyte genes, more than 70% of genes examined were dominantly expressed in germline tissues rather than somatic tissues. This percentage is much higher than seen overall for genes that were not previously called oocyte or sperm without considering motif information (20%), suggesting a functional role of CG-rich motifs in germline expression.

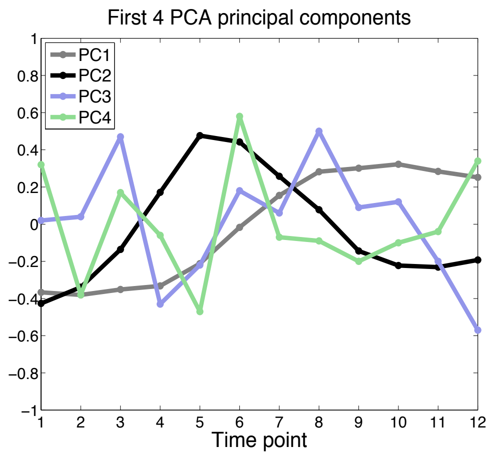
and sperm gene sets Reinke et al. (2004), about 20% of genes that were not identified as oocyte or sperm had the germline expression by *in situ* analysis. Table 3.1 shows that for the genes that were associated with oocyte motifs 1, 2 and 3 via latent factor analysis – but had not previously been identified as oocyte genes – 37/52, 43/55, and 38/62 showed germline expression. All these proportions are very significantly higher than the background percentage of 20% ($p < 8.0\text{e-}16$ for all motifs by a proportions test). These results provide additional evidence that we are learning functional motifs that contribute to germline expression.

Comparison with principal component analysis

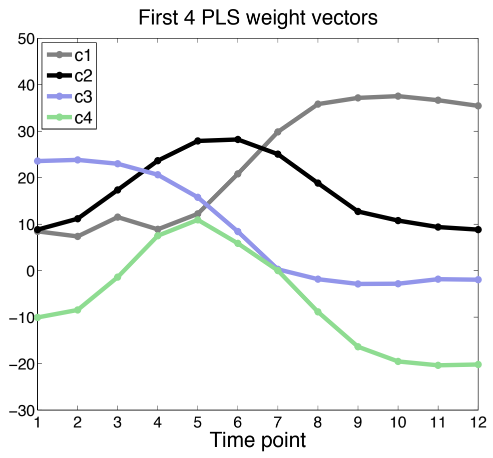
Principal component analysis (PCA) is a widely used dimensionality reduction technique that extracts from the data matrix a sequence of orthogonal vectors, or principal components, that capture the directions of maximal variance in the input data. PCA is frequently used on either rows (genes) or columns (experiments) of a gene expression matrix for visualization or preprocessing prior to other kinds of analysis Raychaudhuri et al. (2000). By contrast, PLS is a supervised method that, in our context, determines weight vectors $\mathbf{c}_i$ as directions in gene expression space having maximal covariance with latent factors in motif space. Both PCA components and PLS weight vectors are interpreted as gene expression patterns. However, principal components are learned from gene expression data only, while weight vectors $\mathbf{c}_i$ are found based on a linear mapping from motif space to gene expression space.

We were interested in comparing our (graph-regularized) PLS results with standard PCA in order to assess the value added by the motif information and supervised learning formulation. We anticipated some concordance of results, since directions that capture little variance in the expression data will also fail to have significant covariance with motif latent factors. Figure 3.12(a) and 3.12(b) plot the first four PCA components versus PLS weight vectors. The first and second PCA components indeed bear some similarity to the first and second PLS weight vectors and to some extent resemble the oocyte and sperm gene expression patterns, respectively. Since these two gene sets are fairly large and follow distinct expression patterns, they account for a significant portion of gene expression variance, and so it is not surprising that the first PCs show correlation with these patterns. However, all the principal components are less smooth, as expression trajectories, than their corresponding PLS weight vectors, and the smoothness of the PCs deteriorates more rapidly than in PLS as the number of principal components/latent factors increases. It therefore appears that PLS uses motif information to provide some degree of regularization on the weight vectors, leading to smoother expression patterns corresponding to latent factors.

To confirm that the PLS-derived motifs could not be determined from anal-



(a)



(b)

Figure 3.12: **Comparison of PCA components and PLS expression weight vectors in gene expression space.** The first and second principal components bear some similarity to corresponding PLS weight vectors $\mathbf{c}_i$, $i = 1, 2$, but all principal components are less smooth than in PLS. (a) PCA identifies the first four directions (PC_1, PC_2, PC_3 and PC_4) that have maximal variance in gene expression space. Principal components are plotted v.s. time. (b) Graph-regularized PLS learns weight vectors ($\mathbf{c}_1$, $\mathbf{c}_2$, $\mathbf{c}_3$ and $\mathbf{c}_4$) based on a linear mapping from motif space to gene expression space. Weight vectors are plotted vs. time.

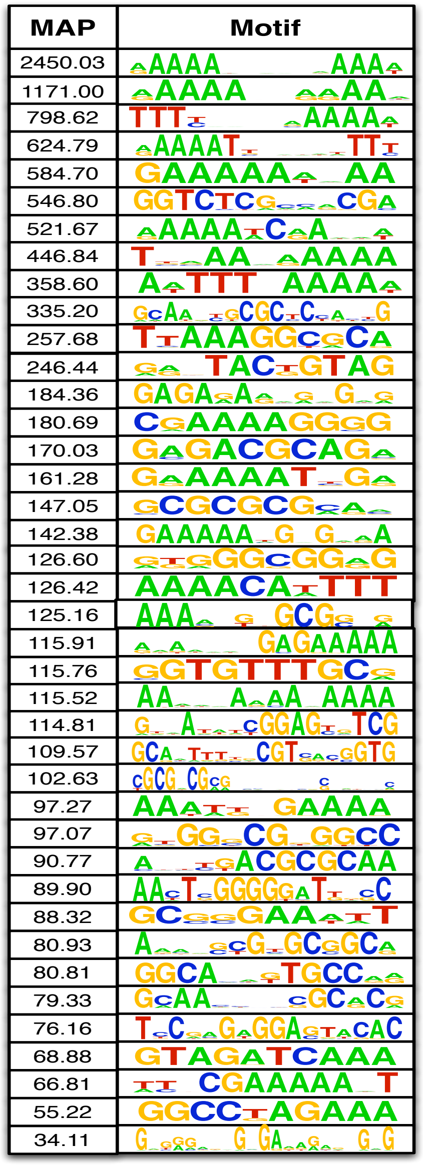
ysis of the first and second principal components (PC_1 and PC_2), we further compared PCA and PLS in terms of extracted motifs. We used AlignACE Hughes et al. (2000a), a Gibbs sampling based motif finding algorithm, to discover motifs associated with the first two PCs, using the following procedure. First, we selected genes highly correlated with PC_1 and PC_2 (Pearson correlation coefficient ≥ 0.9) and obtained two gene sets consisting of 1248 and 415 genes for PC_1 and PC_2 respectively. Second, we ran AlignACE on the upstream regions of genes in each set, producing 58 motifs for PC_1 and 89 motifs for PC_2 in order of descending MAP scores, the metric for motif strength used by AlignACE. Figure 3.13(a,b) shows the two tables consisting of top 40 motifs for PC_1 and PC_2 respectively. In both tables, we see many AA-rich and GG-rich motifs that are highly ranked by MAP score, which likely come from low complexity sequence regions and probably do not represent biological binding sites. In Figure 3.13(a) for PC_1, AlignACE found several CG-rich motifs with relatively low MAP scores (e.g. MAP = 147.05, 90.77, 80.93). Among these motifs, most contain the core element CGCGC, matching the top ranked 50 k-mers of 1st PLS weight vector.

In Figure 3.13(b) for PC_2, only one motif (MAP score = 101.03) is similar to our PLS sperm gene motif ACGTG from the 2nd weight vector. However, it ranks low by MAP score and top ranked motifs seem to be background sequences with local AA or GG enrichment. These results suggest that we cannot fully retrieve the motifs learned by PLS simply by analyzing genes correlated with PC_1 and PC_2. Rather, PLS appears to recover more complete motif information by directly setting up a correspondence between promoter sequence and gene expression.

Since the third and fourth PLS latent factors represent much smoother and quite different expression patterns than their PCS counterparts, we examined whether the genes associated to these factors based on motif and expression similarity (see Methods) may have common functions. While there were few genes associated to the fourth PLS factor (18 genes) showed no enrichment for GO terms, the gene set for the third PLS factor was significantly enriched for 54 GO terms (using a threshold of $p <$ 1e-4, uncorrected hypergeometric people), of which the majority involved metabolism (32/54) and almost half of these were specific to amino acid metabolism (15/54). These genes are not enriched for germline expression, suggesting that our analysis has uncovered an independent co-regulation of a set of gene functions that might have been swamped out by the stronger germline information using other techniques.

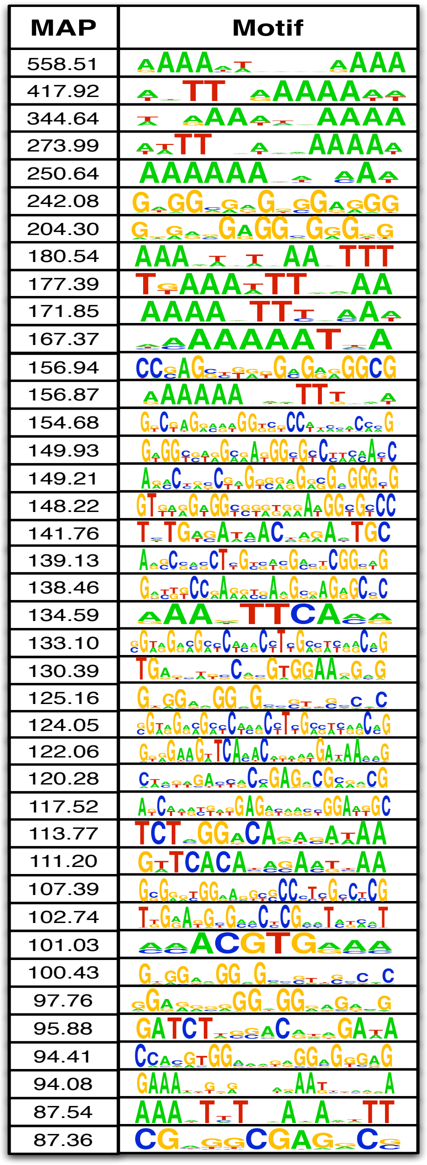
Comparison with clustering

Finally, we compared our results with standard cluster-first analysis, using hierarchical clustering to identify 5 distinct gene clusters and applying the AlignACE



MAP	Motif
2450.03	
1171.00	
798.62	
624.79	
584.70	
546.80	
521.67	
446.84	
358.60	
335.20	
257.68	
246.44	
184.36	
180.69	
170.03	
161.28	
147.05	
142.38	
126.60	
126.42	
125.16	
115.91	
115.76	
115.52	
114.81	
109.57	
102.63	
97.27	
97.07	
90.77	
89.90	
88.32	
80.93	
80.81	
79.33	
76.16	
68.88	
66.81	
55.22	
34.11	

(a)



MAP	Motif
558.51	
417.92	
344.64	
273.99	
250.64	
242.08	
204.30	
180.54	
177.39	
171.85	
167.37	
156.94	
156.87	
154.68	
149.93	
149.21	
148.22	
141.76	
139.13	
138.46	
134.59	
133.10	
130.39	
125.16	
124.05	
122.06	
120.28	
117.52	
113.77	
111.20	
107.39	
102.74	
101.03	
100.43	
97.76	
95.88	
94.41	
94.08	
87.54	
87.36	

(b)

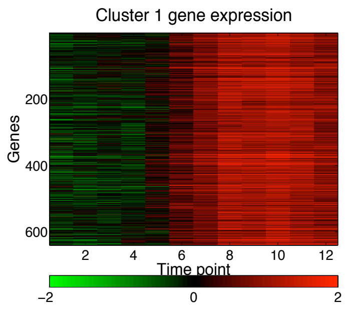
Figure 3.13: **Motifs found by AlignACE in genes correlated with PC_1 and PC_2.** (a) Top 40 AlignACE motifs in genes correlated with PC_1 sorted by MAP score. Top ranked AA-rich and GG-rich motifs may result from low complexity regions, and several PCA motifs with relatively low MAP scores (e.g. MAP = 147.05, 90.77, 80.93) are similar to PLS 1st factor motifs. (b) Top 40 AlignACE motifs in genes correlated with PC_2. Only one motif (MAP score = 101.03) is similar to our PLS sperm gene motif ACGTG from 2nd weight vector. None of the other PCA motifs matched any of the PLS 2nd factor motifs.

motif discovery program to the promoters of each cluster in order to find over-represented motifs. In the hierarchical clustering step, we clustered genes by the similarity of their temporal expression profiles and determined gene clusters with distinct expression patterns. Using average linkage for calculation of cluster distance, we identified five large gene clusters within which the Pearson coefficient exceeds 0.80. In particular, we found three large gene clusters exhibiting expression patterns similar to oocyte or sperm gene expression, as illustrated in Figure 3.14. Genes in Cluster 1 display very low levels of expression early in the time course (time points 1 to 5) and then show an abrupt increase (time points 6 and 7). Meanwhile, genes in Cluster 2 have higher levels of expression at early time points and show a more gradual increase in expression over time. Genes in Cluster 3 are characterized by a sharp increase in expression at time points 3 and 4 and a sharp decline at time points 7 and 8, an expression pattern seen in many sperm genes. We applied AlignACE analysis on the three gene clusters and learned 47, 53 and 36 motifs for Clusters 1, 2 and 3 respectively. Figure 3.14 displays the three motif tables consisting of the top 40 motifs for Clusters 1 and 2 and all 35 motifs for Cluster 3, sorted in order of descending MAP scores. Similar to AlignACE on PCA gene sets, there are many AA-rich and GG-rich motifs that may come from low complexity sequence regions. For Cluster 1 and 2, which resemble the oocyte gene expression patterns, four motifs with relatively low MAP scores (MAP = 87.22, 57.85 with ranks 22, 28 among motifs in Cluster 1; and MAP = 109.32, 95.99 with ranks 24, 24 among motifs in Cluster 2) match PLS 1st weight vector motif CGCGC. Two motifs (MAP = 192.24, 120.51 at ranks 14, 21) in Cluster 2 contain the core element GGCGC found by PLS 1st weight vector. For Cluster 3, none of the AlignACE motifs match the top ones found by PLS 2nd weight vector. We conclude first that PLS avoids many presumably spurious motifs from low complexity regions while finding true germline-specific motifs that are missed through standard cluster-based analysis.

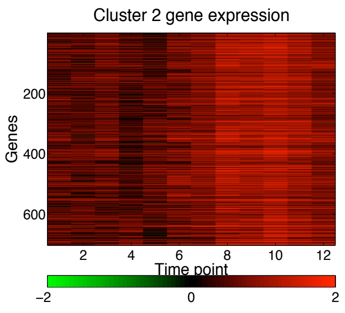
Univariate PLS regression

n multivariate PLS, we found latent factors in motif space that explain gene expression trajectories over all time points simultaneously. For comparison, we also investigated learning motif information to predict time-specific gene expression by applying PLS to single experiment gene expression values, similar to existing regression-based algorithms like REDUCE Bussemaker et al. (2001). We ran standard univariate PLS regression, where we trained and tested each time point separately. We learned up to five latent factors per time point, giving a total of sixty factors. Figure 3.15(a) plots the normalized mean squared error on cross-validation test data versus number of PLS iterations for univariate and multivariate PLS . At each PLS iteration, univariate PLS learns twelve latent factors, corresponding to the twelve time points, while multivariate PLS learns one latent factor for all time points together. As shown in the Figure 3.15(a), univariate PLS achieves its lowest test error at the 1st iteration (12 latent factors), performing similarly though in fact marginally better than the best cross-validation error for standard multivariate PLS at the 4th iteration (4 latent factors). We conclude that learning motifs for each time point indendently does as well as (indeed, slightly better than) the multivariate approach in terms of reducing squared error, but it does so at the cost of a more complex model.

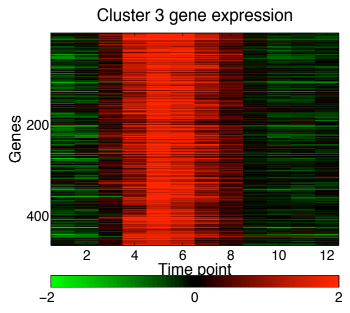
We were also interested in evaluating univariate PLS on biological gene sets across time points, to see whether correlating motifs with certain time points can significantly explain the differential expression of the gene sets. We only examined the time-specific twelve factors corresponding to first latent factors for each time point, as univariate PLS starts overfitting after the 1st iteration. Figure 3.15(b) plots the time-specific normalized mean squared prediction error versus time point



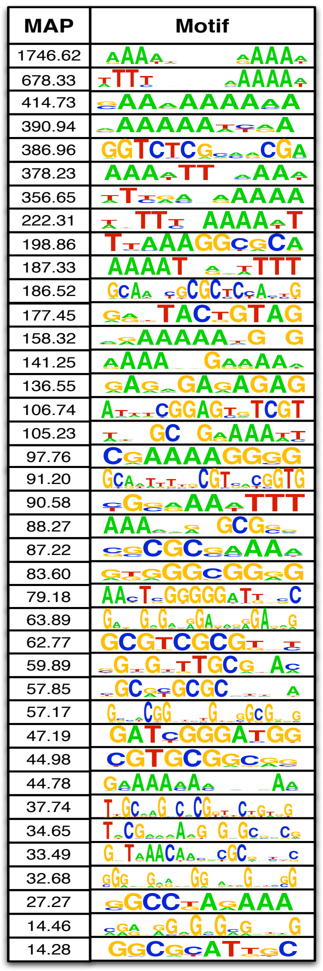
(a)



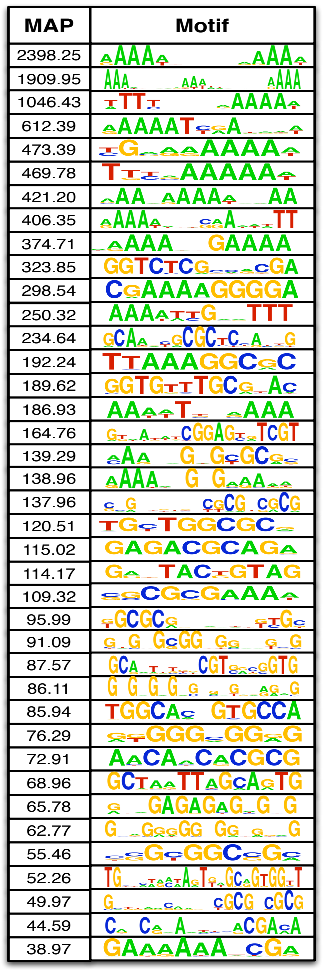
(b)



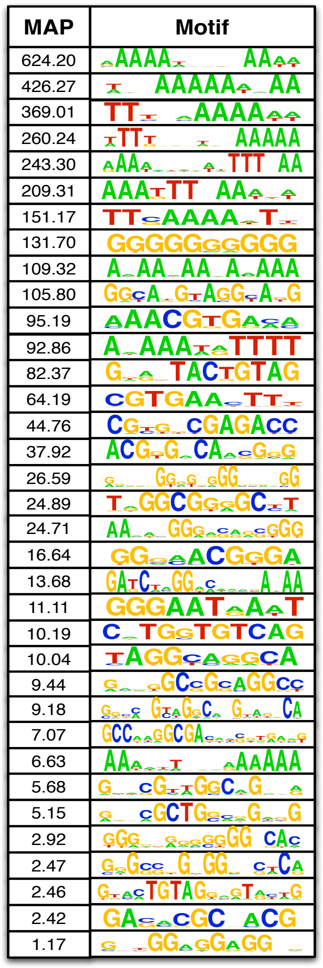
(c)



MAP	Motif
1746.62	AAAA AAAA
678.33	TTT AAAA
414.73	AAAAAAAAA
390.94	AAAAAA A
386.96	GGTCTCG CGA
378.23	AAA TT AA
356.65	T AAAA
222.31	TT AAAA T
198.86	T AAAGGCGCA
187.33	AAAAT TTT
186.52	GCA CGCTCA G
177.45	G TACTGTAG
158.32	AAAAA G G
141.25	AAAA GAAA
136.55	GAG GAGAGAG
106.74	A CGGAG TCGT
105.23	GC GAAAA
97.76	CGAAAAGGG
91.20	GCA CGT CGGTG
90.58	G AA TTT
88.27	AAA GCG
87.22	CGC AAA
83.60	G GGCGG G
79.18	AA T GGGGGAT C
63.89	G G G GA GA G
62.77	GCGTCGCG
59.89	G G TGCG A
57.85	GC GCGC A
57.17	G CGG G GCG G
47.19	GATCGGGATGG
44.98	CGTGCGGC
44.78	GAAAAAA AA
37.74	T GC G C CG C G
34.65	T CG A G GC C
33.49	G TAAACAA CGC C
32.68	GG G GG G G
27.27	GCC A AAA
14.46	G G G G G
14.28	GGCGCAT C



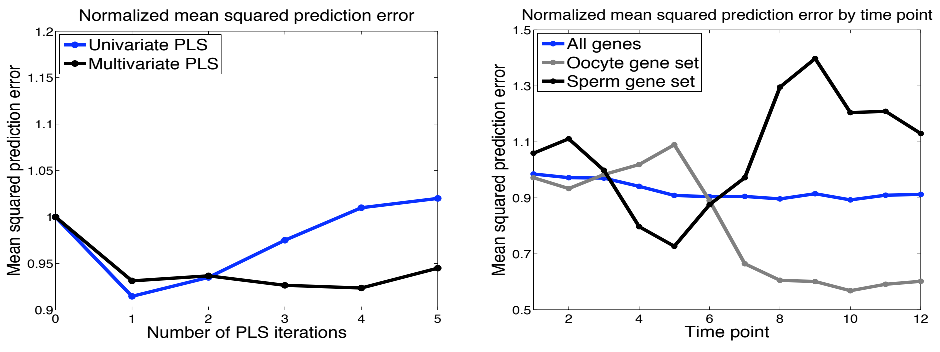
MAP	Motif
2398.25	AAAA AAA
1909.95	AAA AAA AAA
1046.43	TTT AAAA
612.39	AAAAT A
473.39	G AAAAA
469.78	TT AAAAA
421.20	AA AAAA AA
406.35	AAAA CA TT
374.71	AAAA GAAAA
323.85	GGTCTCG CGA
298.54	CGAAAAGGGGA
250.32	AAAA G TTT
234.64	GCA CGCTC A G
192.24	TTAAAGGCGC
189.62	GGTG TGCG AC
186.93	AA T AAA
164.76	G A CGGAG TCGT
139.29	AA G GCGC
138.96	AAAA G GAAA
137.96	C G CGCG CGCG
120.51	TG TGGCGC
115.02	GAGACGCAGA
114.17	GA TACTGTAG
109.32	CGCGAAA
95.99	GCGCG TGC
91.09	G G GCGG GG G G
87.57	GCA CGT GGTG
86.11	G G G G G G A G
85.94	TGGCA GTGCCA
76.29	GGG GG G
72.91	AACA CACGCG
68.96	GCTAA TTAGCAGTG
65.78	G GAGAGAG G G
62.77	G GGGGG GG G G
55.46	GGCGGCGG
52.26	TG A T GCAGTGG T
49.97	G C CGCG CGCG
44.59	C C A ACGACA
38.97	GAAAAAA CGA



MAP	Motif
624.20	AAAAA AA
426.27	T AAAAA AA
369.01	TT AAAAA
260.24	TTT AAAAA
243.30	AAA TTT AA
209.31	AAATTT AA
151.17	TTCAAAA T
131.70	GGGGGGGGG
109.32	A AA AA A AAA
105.80	GGCA GTAGG A G
95.19	AAACGTGA A
92.86	A AAA TTTT
82.37	G TACTGTAG
64.19	CGTGAA TT
44.76	CG G CGAGACC
37.92	ACG G CA GG
26.59	G GG G GG GG
24.89	T GGCGGGCT
24.71	AA GG CGGG
16.64	GG ACGGA
13.68	GATC GG A AA
11.11	GGGAAT AAT
10.19	C TGGTGTCAG
10.04	TAGGCAGGCA
9.44	G GCGCAGGCC
9.18	G GAGC G CA
7.07	GCC GGCGA
6.63	AA T AAAAA
5.68	G CG TGGC G A
5.15	G CGCTG G G
2.92	GGG GGG CAC
2.47	G GCC G GG CTCA
2.46	G CTGTAG TA G
2.42	GA CGC ACG
1.17	G GG GGAGG

(d) (e) (f)

Figure 3.14: **Motifs found by AlignACE in different gene clusters.** (a) Expression patterns of genes in Cluster 1. (b) Expression patterns of genes in Cluster 2. (c) Expression patterns of genes in Cluster 3. (d) Top 40 AlignACE motifs found in Cluster 1 genes. (e) Top 40 AlignACE motifs found in Cluster 2 genes. (f) All 35 AlignACE motifs found in Cluster 3 genes.



(a) (b)

Figure 3.15: **Normalized mean squared prediction error on cross-validation test data.** (a) Normalized mean squared error versus number of PLS iterations for standard univariate and multivariate PLS. At each iteration, standard univariate PLS learns twelve latent factors, corresponding to the twelve individual time points, while multivariate PLS learns one latent factor for all time points. Univariate PLS yielded a slightly lower test error than that of standard multivariate PLS after the 1st iteration; however, after one iteration, the univariate PLS corresponds to a collection of motif sets, each predicting a single experiment's gene expression changes, while multivariate PLS uses a single motif set to predict full gene expression trajectories. (b) Normalized mean squared error on test data by time point after the 1st univariate PLS iteration. Normalized mean squared error versus time point on all genes, oocyte and sperm gene sets. Univariate PLS reaches lowest prediction error on oocyte gene set at late time points when oocyte gene expression peaks. Similarly, prediction error on sperm gene set is small at middle time points when sperm gene expression peaks. Each time-specific univariate PLS models the motif-expression correspondence for the gene set differentially expressed at the given time point.

as the first twelve latent factors were applied to the twelve time points. We also estimated the prediction error on oocyte and sperm gene sets, and found that their prediction error profiles seemed to be anti-correlated with their expression profiles, respectively. Univariate PLS reaches lowest prediction error on oocyte gene set at late time points when oocyte gene expression peaks. Similarly, prediction error on sperm gene set is small at middle time points when sperm gene expression peaks. These results are expected, as each time-specific univariate PLS models the motif-expression correspondence for the gene set differentially expressed at the given time point.

Nonetheless, we found the k-mers ranked top by weight vectors at those middle or late time points to be fairly similar. This redundancy confirmed our earlier hypothesis that neighboring time points, either in the middle or late stages, are correlated and help us discern essentially the same motifs. Multivariate PLS reduces this type of redundancy in the model by learning fewer latent factors to map from motif to full expression patterns.

3.5 Conclusions and Discussion

We present a predictive framework for modeling the natural flow of information, from promoter sequence to expression, to learn *cis* regulatory motifs and characterize gene expression patterns in developmental time courses. We introduce a cluster-free algorithm based on a graph-regularized version of partial least squares (PLS) regression to learn sequence patterns – represented by graphs of k-mers, or "graph-mers" – that predict gene expression trajectories. Applying the approach to wildtype germline development in *Caenorhabditis elegans*, we found that the first and second latent PLS factors mapped to expression profiles for oocyte and sperm genes, respectively. We extracted both known and novel motifs from the graph-mers associated to these germline-specific patterns, including novel CG-rich motifs specific to oocyte genes. We found evidence supporting the functional relevance of these putative regulatory elements through analysis of positional bias, motif conservation and *in situ* gene expression. This study demonstrates that our regression model can learn biologically meaningful latent structure and identify potentially functional motifs from subtle developmental time course expression data.

There have been several other regression based motif discovery approaches related to our work. For example, REDUCE Bussemaker et al. (2001) was the original method to use correlation between k-mers and differential expression for motif discovery. REDUCE, however, uses each experiment independently, where we use multivariate PLS to treat full expression trajectories as the output space. To weight the benefits of regression with a multivariate output, we also tried fitting a separate graph-regularized univariate PLS model on each time point separately. We found that multivariate PLS outperforms univariate PLS, suggesting that correlating motifs with full expression patterns is more statistically accurate than performing regression one experiment at a time, at least in the case of correlated experiments such as time series data. Moreover, there was substantial overlap in the motif information inferred from nearby time points, showing that fitting a separate model for each time point entails a good deal of redundancy.

More recently, Zhang et al. Zhang et al. (2008) used PCA to define a basis of univariate response variables in the output space and then performed a REDUCE-like regression onto each variable to collect a set of motifs. In our work, by doing multivariate regression, we retain more structure in the solution, for example, a stratification of the output space by images of latent factors, each one corresponding to a characteristic time expression profile. We also note that lasso regression has been used elsewhere for learning regulatory networks in bacteria using time course expression data Bonneau et al. (2006), and standard PLS has been used with a collection of known motifs in linear modeling of expression data in yeast and bacteria Brilli et al. (2006). Finally, graph-based motif representations have been used previously by other groups, for example Naughton et al. Naughton et al. (2006), but this work again falls into the "cluster-first" category in that it seeks to find overrepresented motifs for a predefined gene set. By contrast, we learn motifs via a global regression problem, and the graph structure is encoded as a constraint on the solution.

Chapter 4

A Predictive Approach to Learning Regulatory Motifs and Control in *Caenorhabditis elegans*

4.1 Introduction

The ability of individual cell lineages within multicellular organisms to sense and respond to their environment hinges on the coordinated function of thousands of genes and their products. A central computational challenge of the past decade has been revealing the underlying network of causal connections among these genes and products — principally, the protein-DNA interactions of transcription factors and the short sequence elements to which they bind in order to regulate the expression of genes — from noisy and incomplete but high-throughput genomic data such as mRNA expression data from microarray experiments.

Most recent machine learning efforts to study gene regulatory mechanisms at a systems level have focused on learning modular or network structure in gene expression data — for example, finding clusters of potentially co-regulated genes, or building a graph of putative regulatory "edges" between genes. In particular, probabilistic graphical models, also called Bayesian networks Friedman (2004); Segal et al. (2003a); Hartemink et al. (2001); Pe'er et al. (2001; 2002); Beer and Tavazoie (2004) have been widely used for learning structure within a formal probabilistic framework where the conditional dependence relationships between various random variables are constrained by a directed graph. Other authors have tried to learn explicit parameterized models for pieces of the regulatory network by fitting linear models to the training data Yeung et al. (2002); D'Haeseleer et al. (1999). Clustering approaches have of course been widely used in gene expression analysis (e.g. et al. (1998)), along with alternate approaches for revealing modular structure Bergmann et al. (2003). An appealing feature of these structure-oriented approaches is that the models are interpretable and therefore provide useful exploratory tools for generating biological hypotheses about gene regulation. We note that almost all this structure modeling work has been limited to the yeast *S.* cereviseae.

Other efforts have focused on using traditional statistical approaches to find individual regulatory patterns that independently account for differential gene expression. For example, the REDUCE method of Bussemaker et al. Bussemaker et al. (2001) discovers motifs whose presence individually correlates with differential mRNA expression in a single microarray experiment. In other work, Pilpel et al. Pilpel et al. (2001) find "synergistic" pairs of motifs whose joint presence correlates with significantly greater gene expression coherence than occurrence of either motif alone. Such statistical approaches can find strong regulatory signals, but do not allow us to detect or integrate many subtler regulatory effects.

More broadly, most methods for discovering transcription factor binding sites rely on first clustering genes (based on expression profiles, annotations, or both) and then looking for overrepresented patterns in the regulatory sequence for these genes.

Although the numerous "cluster-first" approaches — such as MEME Bailey and Elkan (1994b), Consensus Hertz and Stormo (1999), Gibbs Sampler Lawrence et al. (1993), AlignACE Hughes et al. (2000b) and many others — include methods that use probabilistic models, we still characterize them as signal-finding approaches, since they do not learn integrated gene regulation models.

An alternative computational approach for learning predictive models of gene regulation called "regulatory programs" was previously proposed based on the MEDUSA algorithm Middendorf et al. (2005; 2004). The goals of the approach are twofold. First, we want our gene regulatory programs to explain the context-specific regulation of target genes in terms of meaningful mechanistic information, including the activity of transcriptional regulators and signal transducers and the presence of binding motifs in the promoter sequences that mediate regulatory control. Therefore, rather than directly learning a network or a set of clusters or modules, we are learning a prediction function, and we view the learning task as a prediction problem rather than a model selection problem. Second, in order to provide unambiguous statistical validation, we want our gene regulatory programs to achieve high prediction accuracy on held-out data. In the previous work in yeast, MEDUSA achieves both these goals while still yielding interpretable and experimentally testable biological hypotheses.

While the previous work on MEDUSA and the bulk of the structure-learning methods described above have dealt almost exclusively with the *Saccharomyces cerevisiae* (baker's yeast), here we are interested in studying gene regulation in higher eukaryotes with sequenced genomes in general, but focusing on the most tractable of these organisms, the worm *Caenorhabditis elegans.* There are numerous issues in extending computational learning methods to higher eukaryotes. First, the number of regulatory components (transcription factors, signaling molecules) is much larger and less well elucidated. In general, the regulatory sequence information in the non-coding DNA is larger — regulatory elements in the fly, for example, can be found 10K base pairs up or downstream of the gene or in introns Berman et al. (2002) — and more complex, often requiring the modeling of cis regulatory modules, spatial clusters of binding sites that acts as an irreducible functional element Berman et al. (2002); Rajewsky et al. (2002). In worm, however, the promoter sequences are similar to those in yeast in length and complexity, allowing us to use the MEDUSA's current sequence representation. It is nevertheless true that the knowledge of the binding sites in the regulatory sequence is far more limited and motif discovery approaches less developed in worm. Gene expression data can also be more difficult to interpret in multi-cellular organisms. Ideally pure populations of cells should be isolated and be prepared for gene expression data of individual cell types. However at only 1mm in length as an adult, *Caenorhabditis elegans* makes tissue dissection tedious or even impossible for generating homogeneous tissue for biological analysis. For example, if a microarray experiment is performed on whole embryos in development, the gene expression measurements observed are averaged over all cells in the organism, and spatial patterns of differentiation that are crucial for development are lost.

With MEDUSA we obtain encouraging results both in terms of prediction accuracy and in the biological information we are able to extract from the MEDUSA regulatory program. In particular, we present a case study where we detect a signal of lineage-specific regulation despite the fact that we learn from whole embryo expression data.While we do not address all the modeling challenges of higher eukaryotes in general, the experimental results described below are a significant step towards representing more complex regulatory mechanisms in our predictive modeling approach.

4.2 Methods

The core of our approach is a novel algorithm called MEDUSA Middendorf et al. (2005) (= Motif Element Discrimination Using Sequence Agglomeration), which integrates mRNA expression and regulatory sequence data to discovers motifs representing putative transcription factor binding sites and to build a global gene regulatory program. MEDUSA differs from most previous studies by implementing a number of key algorithmic features: (1) it integrates promoter sequence and expression to learn a global regulatory program; (2) it learns binding site motifs directly from sequence without seeding the algorithm with known motifs; (3) it models functional contributions of both regulators and motifs in the regulation of target genes; (4) it avoids overfitting when training in a high dimensional feature space by use of a machine learning technique called boosting.

The inputs to the MEDUSA algorithm are a list of regulators, including those that do not bind DNA, the promoter sequences for all target genes, and gene expression training data that has been discretized into up, down, and baseline expression levels. MEDUSA learns sequence motifs whose presence in the promoters of target genes, together with the mRNA levels of regulators across experimental conditions, helps to predict the differential (up/down) expression of the targets. MEDUSA uses boosting Freund and Schapire (1997), a general binary prediction algorithm from statistical learning theory, to build this prediction function or regulatory program. Empirically, boosting often learns to make large-margin (confident) predictions on the training set, which is theoretically linked to its ability to obtain good generalization on test data even when the feature space is very high dimensional (that is, it avoids overfitting the training data).

MEDUSA models the control logic of transcriptional regulation in the form of an alternating decision tree (ADT). An ADT is a generalization of a decision tree that consists of alternating layers of decision nodes, which ask yes/no questions based on particular features, and prediction nodes, which contain a real-valued score associated with the yes or no answer. Given the promoter sequence of a gene and the expression level of the regulators in an experiment, the MEDUSA regulatory program asks yes/no questions of the form, "Is motif X present in the upstream region of the gene and is the state of regulator Y up (or down) in that experiment?", in the ADT decision nodes. If the answer is "yes", we add the real value contained in the prediction node to the overall prediction score for the example, and we continue down to the next decision node; if the answer is "no", there is no score contribution. To compute the prediction score for a gene-experiment example, we start at the root node and recursively check which decision nodes we can pass through by answering "yes" to the condition, working from the top to the bottom of the ADT; the prediction score is the sum of all the prediction node scores in all paths in the ADT that we visit in this process. Figure 4.1 summarizes the way in which MEDUSA represents the training data and how the learned ADT defines a genome-wide regulatory program.

4.3 Data Set

We performed MEDUSA experiments on a gene expression data set for embryonic development in the worm *C. elegans* Baugh et al. (2003). The data set consists of a finely sampled time course that commences with the zygote and extends into midgastrulation, spanning the transition from maternal to embryonic control of development and including the presumptive specification of most major cell fates. The data contain 7 time points with multiple replicates for each experiment. The data were transformed to fold changes using the PC32 time point (32 minutes after pseudocleavage) as control. The replicate data was used to estimate an intensity-dependent noise model. We discretized the expression data into 3 levels (up, down and baseline) using an intensity-dependent noise model: +1(-1) representing sig-

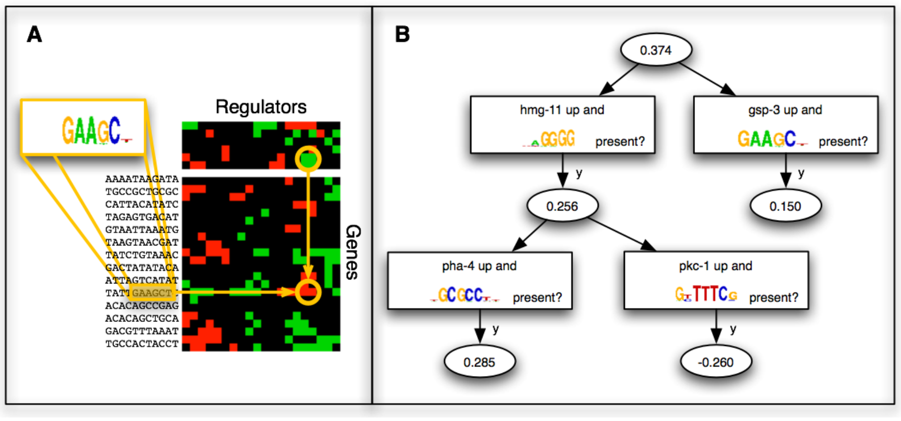


Figure 4.1: **MEDUSA learns genome-wide, context-specific regulation programs.** A schematic example shows how MEDUSA regulatory programs predict differential target gene expression. (A) In the data representation, rows represent genes and columns represent experiments. Genes are divided into regulators (transcriptional regulators and signal transducers) and targets. The expression levels of regulators, along with the promoter sequences of target genes, are used to predict up/down expression of the targets. Individual weak rules in MEDUSA depend on a pairing of a particular regulator state and a motif. For example, the rule illustrated suggests that a certain regulator is in a down state (low expression level), targets genes containing the motif "GAAGCT" in their promoters tend to be upregulated; while a single gene-experiment example is highlighted, the weak rule must be predictive across the (weighted) training data in order to be chosen. (B) The MEDUSA regulatory program is described by an alternating decision tree that asks questions about the expression level of regulators in the experimental condition and the presence of motifs in the gene's promoter. Using boosting, weak rules are iteratively added as nodes in the ADT; the scores in each round node indicate the contribution to the overall prediction score when the corresponding weak rule applies. The ADT can be applied to a new gene-experiment example to obtain a real-valued prediction score. The sign of the score gives the up/down prediction, while the size of the score is a measure of its confidence.

nificant up-(down-)regulation and 0 representing expression measurements within the level of noise using a p-value of 0.01) and obtained a total of 9135 genes that significantly changed expression in at least one time point.

We collected a set of 1370 potential regulators consisting of transcription factors, kinases, phosphates and signaling molecules from TRANSFAC Matys et al. (2006), WormBook,(http://www.wormbook.org) and WormPD Costanzo et al. (2000). We obtained promoter sequences spanning 1000 bp upstream of the genes from Wormmart (http://www.wormbase.org/biomart/martview). These data were used as input to the MEDUSA algorithm.

4.4 Results

4.4.1 Statistical Validation

4.4.1.1 MEDUSA predicts held-out experimental data without overfitting

A central goal of machine learning is to avoid overfitting, i.e. to ensure that, as the complexity of the learned model increases, the model continues to generalize well to new data drawn from the same distribution as the training data. To confirm that MEDUSA's learned regulatory program generalizes well, we performed 10-fold cross-validation experiments, randomly splitting gene-experiment examples into test and training sets with 10% of the data assigned to a test fold. Replicate examples were groups within folds to avoid making the learning task too easy. Figure 4.2 illustrates that after 500 iterations of boosting (i.e. 500 weak rules combined in the ADT), the model continues to generalize well to test data.

4.4.1.2 MEDUSA's accuracy is statistically significant

To assess the difficulty of the prediction task, we compared MEDUSA to a simple correlation-based method, where we predict a gene's held-out expression levels based on the "nearest regulator" to its training set expression levels. As in our main experiments, we performed 10-fold cross-validation, and for each gene represented in the test set, we considered its expression profile when restricted to examples (i.e. experiments) in the training set and found the best-correlated regulator across these experiments. The expression level of this regulator was then used to predict up/down expression in experiments held out for this gene. As similarity metrics,

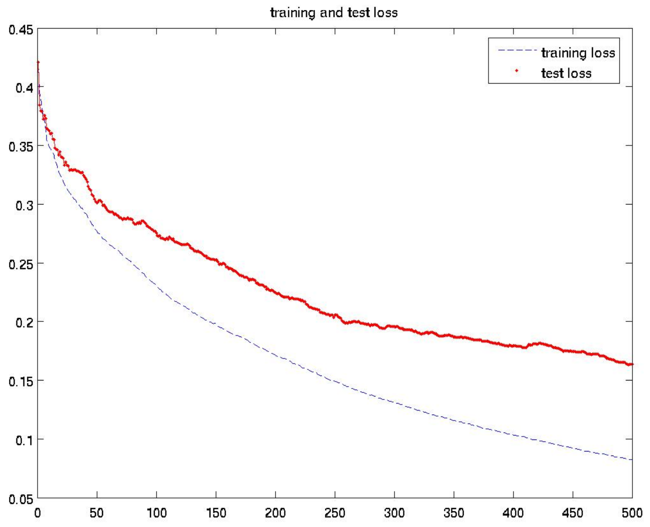


Figure 4.2: **Prediction accuracy on unseen experimental data.** Training and test set error rates for the first 500 rounds of boosting, showing MEDUSA's accurate prediction on test data (red line) as well as agreement with training data (blue line).

we tried both the Pearson correlation over real-valued expression data (including baseline examples) and the normalized Hamming distance (excluding baseline examples) for discretized expression data, where the inclusion/exclusion of baseline examples was chosen in order to report the better results. In cases where multiple regulators were equally distant from a target gene, we randomly selected one amongst them as the "nearest regulator." The ratio of negative to positive examples for the target genes was 42% to 58%; a classifier that always predicts the larger class would therefore achieve the baseline ("random") performance of 58% accuracy. As shown in Table 4.1, we found that MEDUSA greatly outperformed the nearest-regulator method for both experiments using real-valued expression data and for discretized data.

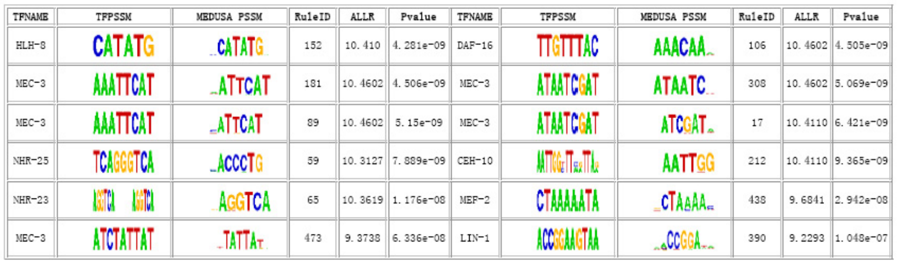
Table 4.1: **Assessing statistical significance of accuracy of predictions on held out experimental data**.

method	expression data	averaged cross-validation accuracy
nearest regulator	continuous	66%
nearest regulator	discrete	73%
MEDUSA	discrete	84%

4.4.2 Biological Validation

4.4.2.1 MEDUSA learns regulatory sequence elements *ab intio*

We compared the MEDUSA PSSMs learned in the first 500 boosting rounds against TRANSFAC and WormBook PSSMs. Figure 4.3 shows the most significant matches of PSSMs and corresponding p-values (calculated from average log likelihood ratios using `MatAlign` (http://ural.wustl.edu/software.html). In particular, we found the binding site for HLH-8, a helix-loop-helix transcription factor expressed in all body wall muscle cells from several cell lineages during embryogenesis, and for MEC-3, a transcription factor essential to touch cell differentiation in the neural lineage.



TFNAME	TFPSSM	MEDUSA PSSM	RuleID	ALLR	Pvalue	TFNAME	TFPSSM	MEDUSA PSSM	RuleID	ALLR	Pvalue
HLH-8	CATATG	CATATG	152	10. 410	4. 281e-09	DAF-16	TTGTTTAC	AAACAA	106	10. 4602	4. 505e-09
MEC-3	AAATTCAT	ATTCAT	181	10. 4602	4. 506e-09	MEC-3	ATAATCGAT	ATAATC	308	10. 4602	5. 069e-09
MEC-3	AAATTCAT	ATTCAT	89	10. 4602	5. 15e-09	MEC-3	ATAATCGAT	ATCGAT	17	10. 4110	6. 421e-09
NHR-25	TCAGGGTCA	ACCCTG	59	10. 3127	7. 889e-09	CEH-10	[illegible]	AATTGG	212	10. 4110	9. 365e-09
NHR-23	[illegible]	AGGTCA	65	10. 3619	1. 176e-08	MEF-2	CTAAAAATA	CTAAAA	438	9. 6841	2. 942e-08
MEC-3	ATCTATTAT	TATTA	473	9. 3738	6. 336e-08	LIN-1	ACCGGAAGTAA	CCGGA	390	9. 2293	1. 048e-07

Figure 4.3: **Significant TF binding site motifs learned by MEDUSA for the worm data set.** The table shows some of the PSSMs found by MEDUSA that most significantly match experimentally verified TF binding sites compiled from the TRANSFAC and WormBase databases. The significance of the match is reported as a p-value for the average log likelihood ratio (uncorrected for multiple hypothesis testing).

4.4.2.2 MEDUSA reveals context-specific regulation relevant to touch receptor neurons in worm

We further investigated whether we could reveal target, context, and even cell lineage specific regulation by examination of the learned MEDUSA regulatory program, despite the limitation that the expression data came from whole embryo samples. We performed a case study relevant to touch receptor neurons. Six mechanosensory neurons (the touch cells) mediate the response of *C. elegans* to gentle touch. Experimental evidence suggests that the gene MEC-3 encodes a transcription factor which specifies the differentiation of the touch cells, Way and Chalfie (1988); Xue et al. (1992); Zhang et al. (2002). and a subset of 34 genes in our data set have been previously identified as MEC-3-dependent genes expressed in touch cells Zhang et al. (2002). We first analyzed this set of genes across all time points after the 4-cell stage during embryonic development in order to find MEDUSA motifs which strongly affect this group of targets. We ranked the motifs using a margin-based score and frequency score. The frequency score is simply the number of training examples that are affected by the motif according to the regulatory program. The margin-based score assesses how much the motif affects the confidence of predictions on a set of training examples.

In large-margin techniques like boosting and SVMs, the margin for an example x_i with label $y_i = \pm 1$ and prediction function f is given by $y_i f(x_i)$. If the margin is positive, the prediction is correct, and the size of the margin gives a measure of confidence in the prediction. If we remove, for example, motif m from the regulatory program (i.e. delete nodes containing m and their subtrees from the ADT learned by boosting), we denote f^{-m} as the modified prediction function and define the following score:

$$S_m = \frac{1}{|T|} \sum_{\{(x_i,y_i)\}\in T} y_i(f(x_i) - f^{-m}(x_i)).$$

Here T is a set of training examples considered. The score S_m will again be positive if on average m is important for making predictions, and its size measures its importance to the target set.

We found a MEC-3 binding site (ATCGAT) among the top scoring motifs using both rank analyses. We also studied two special time points, 53 and 83 minutes after the 4-cell stage, at which time MEC-3 is most up-regulated and potentially most active. In both cases, the same binding site scored the highest among all motifs. In this way, MEDUSA successfully discovered a MEC-3 motif despite the lineage-specific nature of touch cell differentiation. Figure 4.4 illustrates the results of margin score analysis for target genes relevant to touch cell differentiation.

4.5 Conclusions and Discussion

It is useful to emphasize some aspects of what MEDUSA has been able to accomplish in these experiments which suggest promising next steps for the predictive modeling framework. In whole embryo worm expression data, the MEDUSA model can be validated both biologically and statistically, and is predictive both in that it generates hypotheses (provides a ranking of most important associations between transcription factors and regulatory sequences) and in that it makes quantitative predictions of data from completely held-out experiments with statitically significant accuracy. By representing the regulatory control logic of the organism as an alternating decision tree, the model is also highly interpretable: one tree describes the transcriptional regulatory program of the organism for all genes, during all experiments. We also illustrate how MEDUSA can be used to reveal context-specific and lineage-specific regulation relevant to a particular biological behavior of interest (illustrated with a cell lineage-specific mechanism of touch receptor neurons), and can be used to reveal regulatory cascades (the regulation of transcription factors by other transcription factors).

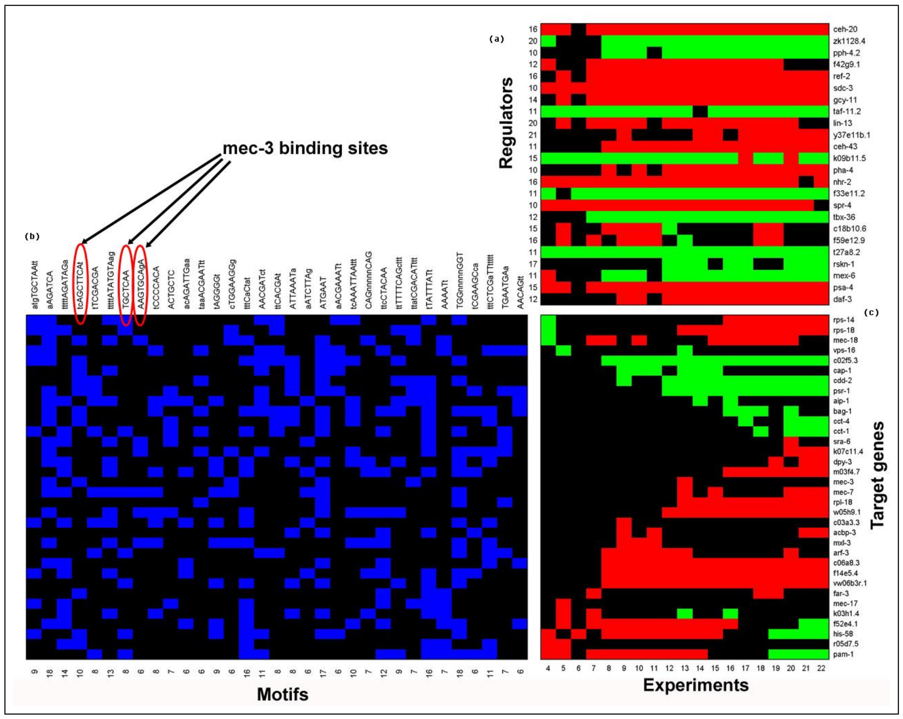


Figure 4.4: **Context-specific regulation for target genes relevant to touch cell differentiation.** (a) Patterns of up (red), down (green), and baseline (black) expression levels for the statistically significant regulators controlling target genes regulated by MEC-3 across the time points after 4-cell stage. At the left of each row, the number of target genes affected by the regulator in these experiments is given. (b) The top-ranked sequence features learned by MEDUSA, as determined by a margin-based score, and their hits across the set of target gene promoters. The PSSMs learned by MEDUSA are represented by their consensus sequences. At the bottom of each column, the number of target genes containing the motif is given. (c) Patterns of discretized gene expression levels for the target genes regulated by MEC-3 across the time points after 4-cell stage.

We note that MEDUSA is able to learn the regulatory binding sites for worm *ab initio*, without any initial guesses as to the binding site information (e.g. by using the expression data first to cluster the genes and then look for sequence elements which correlate with cluster assignment). This suggests that MEDUSA should be able to learn these regulatory elements for eukaryotes of similar or less complexity, e.g., members of the worm or yeast family which have been sequenced but whose regulatory elements have not yet been annotated. Note that while regulatory information learned by MEDUSA may be presented as a network, in common with many approaches for understanding transcriptional regulatory programs, this network is extracted from a quantitative prediction function rather than simply a qualitative edge list. In this way, MEDUSA combines the statistical performance of a modern large margin prediction technique with the ability to visualize and interpret the model through a rendered network.

Although the ability of MEDUSA to repeat in *C. elegans* its successes in *Saccharomyces Cerevisiae* is encouraging, it remains to be seen to what extent such the predictive modeling framework can be used in higher eukaryotes. In *D. melanogaster*, *M. musculus* and *H. sapiens*, the regulatory sequence information in the non-coding DNA is known to be far more complex: unlike in *Saccharomyces Cerevisiae*, where the promoter is limited to a 1000 base pair region directly upstream of the transcription start site, regulatory elements in the *D. melanogaster*, for example, can be found 10K base pairs up or downstream of the gene or in introns Berman et al. (2002). Furthermore, combinatorial regulation is known to be more prevalent and important in higher organisms, and one must deal with the phenomenon of the cis regulatory module, a spatial cluster of binding sites that acts as an irreducible functional element Berman et al. (2002); Rajewsky et al. (2002). Modification of chromatin structure Widom (1998) through histone-DNA interactions in the nucleosomes, histone tail interactions in the chromatin fiber, and the activity of chromatin remodeling complexes represents another set of mechanisms regulating transcription. Finally, numerous post-transcriptional regulatory processes, including mechanisms related to alternative splicing and RNA stability, play an important role in higher organisms.

As we anticipate extending MEDUSA to more difficult multicellular organisms, we note several reasons for optimism and opportunities for carefully designed machine learning approaches. One promising avenue is learning regulatory programs in multicellular eukaryotes from data other than microarray data, which necessarily involves the loss of any spatial or single-cell-specific information. The most notable example of this is the wealth of image data being generated and made publicly available, e.g., in single *D. melanogaster* embryos Janssens, Hou, Jaeger, Kim, Myasnikova, Sharp, and Reinitz (Janssens et al.); Perkins et al. (2006); Grumbling and Strelets (2006). A second specific promising advance is algorithmic development allowing richer representation of sequence data, particularly cis-regulatory modules and conservation information. A cis regulatory module is a spatial cluster of binding sites that acts as an irreducible functional element Berman et al. (2002); Rajewsky et al. (2002); incorporation of spatial information beyond the simple 'presence/absence' representation of MEDUSA will be necessary to learn such structure. Conservation information can be obtained owing to the growing number of sequenced genomes for comparative approaches and in databases of known interactions (e.g. KEGG, MIPS) and gene annotations (e.g. Gene Ontology). For instance, comparisons between non-coding regions of *C. elegans* and *C. briggsae* have revealed important cis-acting regulatory elements controlling gene expression in *C. elegans* Natarajan et al. (2004); Culetto et al. (1999); Teng et al. (2004). Given a known binding site, programs have be used to identify target genes using phylogenetic footprinting Bigelow et al. (2004).

Chapter 5

A comparative study on boosting algorithms for motif discovery in *Saccharomyces cerevisiae*

5.1 Introduction

One of the central challenges in computational biology is learning the structure and control of transcriptional regulatory networks from functional genomic data. The problem of identifying transcription factor-target gene regulatory interactions and the DNA sequence elements that mediate these interactions is a key component in those computational studies. MEDUSA Middendorf et al. (2005; 2004) has been previously presented as a machine learning approach for learning motif models of transcription factor binding sites and gene regulatory programs. By incorporating promoter sequence and gene expression data, MEDUSA learns motifs whose presence in the promoter region of a gene, combined with activity of a regulator in an experiment, that are predictive of differential gene expression.

MEDUSA is based on a popular boosting algorithm: Adaptive Boosting

(AdaBoost), proposed by Freund and Schapire Freund and Schapire (1997). In the MEDUSA model, the role of boosting is to combine many roughly accurate weak rules, associated with motifs and regulatory activity, into one highly accurate prediction function for differential gene expression. In boosting algorithms there are two sets of weights: the weights on training examples and weights on the currently selected weak rules. It has been shown that to determine the weights on the weak rules, AdaBoost is a gradient descent procedure that minimizes an exponential classification error function Freund and Schapire (1997). The performance of a weak rule with respect to weight distribution can be measured by *edge*, which is connected with the weighted classification error. Large *edge* indicates low classification error for a new weak rule. To "decorrelate " new weak rule with last one, AdaBoost updates weights on examples such that the *edge* of the last weak rule with respect to the weight distribution is minimized Kivinen and Warmuth (1999). AdaBoost is very successful as it has been empirically observed that AdaBoost rarely overfits even after the training error reaches zero. This observation has been explained in terms of margins of the training set, where the margin is interpreted as the confidence in the prediction Schapire (1999). AdaBoost is empirically found to improve the margins even after many boosting iterations, which often translates into better performance on the test set.

Following the logic of large-margin theory, alternative boosting algorithms have been proposed that provably optimize margins, e.g. Linear Programming Boosting (LPBoost) Demiriz et al. (2002) and Totally Corrective Boosting (TotalBoost) Warmuth et al. (2006). Both LPBoost and TotalBoost aim to find the optimal linear combinations of weak rules that maximize hard or soft margin on the training set Demiriz et al. (2002). This margin optimization problem has been formulated as a linear programming (LP) problem Grove and Schuurmans (1998). In LPBoost, the dual of this LP leads to a new example reweighting procedure:

at each boosting iteration, the example weight distribution is updated so that the *edges* of all past weak rules w.r.t. weight distribution are constrained. In TotalBoost, the example reweighting procedure minimizes the relative entropy to the initial example weight distribution subject to linear constraints on the *edges* of all selected weak rules. Hence, both LPBoost and TotalBoost are *totally corrective* boosting algorithms in the sense that they optimize weights on examples based on all past weak rules. In contrast, AdaBoost is *corrective* as it only updates the weights on the examples based on the last weak rule.

Theoretically, LPBoost and TotalBoost should have better generalization performance than AdaBoost according to the margin theory Schapire (1999). Because *totally corrective* boosting algorithms directly optimize margins while AdaBoost, at least in some cases, does not Mukherjee et al. (2011). In this paper, we implement both LPBoost and TotalBoost in the MEDUSA model. We run MEDUSA experiments on the dataset of hypoxia response in *Saccharomyces cerevisiae* Lai et al. (2005; 2006) comparing the performance of hard-margin LPBoost, soft-margin LPBoost, soft-margin TotalBoost and AdaBoost. We find there is no statistically significant difference between AdaBoost and soft-margin TotalBoost in their generalization error, but they both outperform hard-margin and soft-margin LPBoost on the hypoxia dataset. We explain the results by considering the margin and weight distribution of training examples. In addition, we perform a comprehensive comparison of the motif discovery results of AdaBoost and TotalBoost. Our findings reveal that TotalBoost identifies many biologically meaningful binding site motifs that are missed by AdaBoost. For motifs found by both TotalBoost and AdaBoost, TotalBoost tends to learn them earlier than AdaBoost, making TotalBoost a more efficient boosting algorithm for extracting biological features in the MEDUSA model.

5.2 Methods

5.2.1 Boosting in MEDUSA

The inputs to the MEDUSA algorithm are a list of regulators and promoter sequences for all target genes, and gene expression training data that has been discretized into up, down and base-line expression levels. MEDUSA learns sequence motifs whose presence in the promoters of targets genes, together with the mRNA levels of regulators across experimental conditions, helps to predict differential gene expression (up/down) Middendorf et al. (2005) .

MEDUSA implements boosting algorithm using the structure of an alternating decision tree (ADT) Freund and Mason (1999a). MEDUSA asks yes/no questions of the form, "Is motif X present in the upstream region of the gene and is the state of regulator Y up (or down) in that experiment?", in the ADT decision nodes. If the answer is "yes", we add the real value contained in the prediction node to the overall prediction score for the example, and we continue down to the next decision node; the prediction score for a gene-experiment example is the sum of all the prediction node scores in all paths in the ADT that we visit in this process.

In the boosting setting, we have a set of labeled examples $(x_{ge}, y_{ge})_{1 \leq e \leq E, 1 \leq g \leq G}$, where g and e are gene and experiment indices and labels y_{ge} lie in $\{-1, +1\}$. The boosting algorithm repeatedly calls a weak rule for a number of iterations $t = 1 \ldots T$ and combines those moderately inaccurate weak rules h^t into a single, highly accurate prediction rule. Boosting maintains a weight distribution $\mathbf{w}$ over the training set denoted by w^t_{ge}. At each iteration t, misclassified examples are assigned higher weights so that the next weak rule will focus more on the hard examples in the training set.

We can measure the performance of a weak rule h^t with respect to weight distribution $\mathbf{w}$ by *edge*: $\gamma_t = \sum_{ge} w^{t+1}_{ge} y_{ge} h_t(x_{ge})$. The *edge* is connected with the weighted classification error ε^t of weak rule h^t: $\varepsilon^t = \frac{1}{2} - \frac{1}{2}\gamma_t$. Large *edge* indicates low classification error for a new weak rule. e.g, a weak rule that predicts perfectly has an *edge* $\gamma = 1$ and a completely random one has a zero *edge*. After a new weak rule is selected, boosting algorithms generally update the weight distribution $\mathbf{w}$ by constraining the *edge* of the new weak rule w.r.t $\mathbf{w}$. This is known as the *corrective* property of a boosting algorithm, e.g. AdaBoost. Some boosting algorithms are *totally corrective*, e.g. LPBoost and TotalBoost, which update the weight distribution $\mathbf{w}$ by constraining the *edge* of all selected weak rules instead of the new one only w.r.t. $\mathbf{w}$.

The final prediction rule of the boosting algorithm is a convex combination of weak rules $F(x_{ge}) = \sum_{t=1}^{T} \alpha_t h_t(x_{ge})$ Freund et al. (1999), where α_t is the coefficient of weak rule at iteration t. The *margin* of a labeled example (x_{ge}, y_{ge}) is defined as $\rho_{ge} = y_{ge}F(x_{ge})$ and the *margin* of the training set is the minimum of example *margins*. Larger *margin* indicates lower error rate on the training set and higher prediction accuracy on the test set.

AdaBoost

AdaBoost was previously implemented for MEDUSA Middendorf et al. (2005; 2004). AdaBoost constructs an additive combination of weak rules by minimizing the exponential loss function:

$$L = \sum_{ge} exp(-y_{ge}F(x_{ge})) = \sum_{ge} exp(-y_{ge} \sum_{t} \alpha_t h_t(x_{ge})) \tag{5.1}$$

AdaBoost can be viewed as a gradient descent method to minimize the exponential loss function. At each boosting round Adaboost selects a new weak rule h^t that classifies training examples with minimal weighted classification error. The linear coefficient of the new weak rule α_t is determined by the weighted classification error. AdaBoost updates the example weight distribution w^{t+1} to increase the weight of misclassified training examples after each boosting round:

$$w_{ge}^{t+1} \propto w_{ge}^{t} exp(-\alpha_t y_{ge} h_t(x_{ge})) \tag{5.2}$$

Further details of the AdaBoost algorithm in MEDUSA could be found in Middendorf et al. (2005; 2004).

5.2.2 LPBoost

Previous studies have shown that AdaBoost is very successful for MEDUSA as its classification accuracy is high Middendorf et al. (2005; 2004). However, AdaBoost is not theoretically perfect from the viewpoint of optimization. In contrast to AdaBoost, LPBoost directly optimizes a margin cost function. The goal of LPBoost is to find the optimal linear coefficients of weak rules $\alpha_1, ..., \alpha_t$ that maximize the minimum margin among the training examples Demiriz et al. (2002). Let the matrix H be a t by N matrix of labelings of the training examples, where t is number of weak rules and N number of training examples. Specifically, $H(x_{ge}) = [h_1(x_{ge}), ..., h_t(x_{ge})]$ is vector of labels given by weak rule $h_1, ..., h_t$ on the training example x_{ge}. LPBoost is formulated as the following LP problem:

$$\boxed{\begin{aligned} max_{\alpha,\xi,\rho} \quad & \rho - D\sum_{ge} \xi_{ge} \\ s.t. \quad & y_{ge} H(x_{ge})\boldsymbol{\alpha} + \xi_{ge} \geq \rho \\ & \xi_{ge} \geq 0, g = 1 \dots G, e = 1 \dots E \\ & \sum_{q=1}^{t} \alpha_q = 1, \alpha_q \geq 0, q = 1, ..., t \end{aligned} \qquad (5.3)}$$

where $\rho - D\sum_{ge}\xi_{ge}$ is the soft margin to be optimized. ξ_{ge} is the slack variable for each training example. D is the tradeoff parameter that balances margin maximization and training error. This LP formulation is also known as ν-LP is with $D = \frac{1}{N\nu}(\frac{1}{N} \leq \nu \leq 1)$, as ν is interpreted as the percentage of training examples misclassified Ratsch et al. (1999). Without D and slack variables ξ, equation (5.4) is the hard-margin version of LPBoost Grove and Schuurmans (1998).

The dual of the LP problem is:

$$\boxed{\begin{aligned} & min_{w^{t+1},\gamma} \quad \gamma \\ & s.t. \qquad \sum_{ge} y_{ge}h^q(x_{ge})w_{ge}^{t+1} \leq \gamma \quad q = 1\ldots t \\ & \qquad\qquad \sum_{ge} w_{ge}^{t+1} = 1, 0 \leq w_{ge}^{t+1} \leq D \end{aligned}} \tag{5.4}$$

The dual LP has a natural interpretation. w^{t+1} is viewed as a probability distribution over the training examples. Thus, $\sum_{ge} y_{ge}h^q(x_{ge})w_{ge}^{t+1}$ is simply the weighted *edge* achieved by h^q on the training set. The best weak rule has a *edge* of γ, and the objective of dual LP is to find a reweighting of the training set, such that the *edge* γ of the best weak rule is as small as possible. Therefore LPBoost is *totally corrective* in the sense that it optimizes the weights based on all past weak rules. In contrast, AdaBoost only updates the weights based on the last weak rule.

Since the matrix H has a very large number of columns, the LP problem (3) has vastly more variables than constraints. The idea of solving the LPBoost formulation is intractable using standard LP techniques because the LP problem has vastly more variables than constraints. The classic column generation (CG) based simplex algorithm Luenberger and Ye (2008) has been successfully applied to the LP problem Bennett et al. (2000); Demiriz et al. (2002). The simplex method avoids considering all variables of an optimization problem explicitly. At each iteration, the CG based simplex algorithm only uses a small subset of the entire variable set, i.e., a subset of the columns of H to learn the current solution. The simplex method first determines if the current solution is optimal, and if it is not, the algorithm will continue to generate some column that will improve the solution. A detailed pseudo-code description of LPBoost implementation in MEDUSA is given in Table 1.

5.2.3 TotalBoost

LPBoost is the most straightforward boosting algorithm for maximizing the soft margin of the linear combination of weak rules. It does so by solving a linear programming problem. In the dual LP, the LPBoost algorithm maximizes edge γ without placing any specific constraints on example weights. One challenge of this is that the weights computed by the simplex method are often sparse Bennett et al. (2000); Demiriz et al. (2002); Warmuth et al. (2006). Hence it could easily happen that LPBoost is "blind" on certain examples when selecting new weak rule. TotalBoost is also a "totally corrective" boosting algorithm that constrains the edges of all past weak rules Warmuth et al. (2006). The difference between LPBoost and TotalBoost is that TotalBoost introduces an entropic regularization on the weights, in order to improve the stability of the boosting procedure.

TotalBoost is motivated by the minimum relative entropy principle: among all the weight solutions that satisfy linear constraint on edge γ, it chooses the weight distribution w^{t+1} that is 'closest' to the initial distribution w^0. The 'closeness' is measured by the relative entropy between distribution w and w^0 defined as follows: $\triangle(w, w^0) = \sum_{ge} w_{ge} \log \frac{w_{ge}}{w^0_{ge}}$. The modified optimization problem is defined in equation (5), where TotalBoost minimizes the relative entropy subject to the constraint that edges of all past weak rules are less than or equal to zero. A detailed pseudo-code description of TotalBoost implementation in MEDUSA is given in figure 5.1.

$$\boxed{\begin{aligned} w^{t+1} = \quad & argmin_w \triangle(w, w^0) = argmin_w \sum_{ge} w_{ge} \log \frac{w_{ge}}{w^0_{ge}} \\ s.t. \quad & \sum_{ge} w_{ge} y_{ge} h^q(x_{ge}) \leq 0 \quad q = 1 \ldots t \\ & \sum_{ge} w_{ge} = 1 \end{aligned}} \tag{5.5}$$

In LPBoost, the upper bound γ on the edge is chosen to be as small as possible, whereas in TotalBoost the weights are chosen to be closest to initial weights as long as γ is less than zero. Therefore in TotalBoost γ decreases more moderately and weights are updated in a more smooth way. Since the initial weight w^0 is uniform, it is unlikely that TotalBoost produces problematic sparse weight distribution during later boosting iterations. In addition, the relative entropy term makes the objective function in (5.5) strictly convex and therefore the optimization problem has a unique solution, which we denote as w^{t+1}. We implement *sequential quadratic programming* (SQP) algorithm Nocedal and Wright (1999) for solving the convex optimization problem. We initially set the approximate solution to $\hat{w} = w^0$ and optimize $\hat{w}$ in a sequential way. Given the current solution $\hat{w}$, we determine the update w by minimizing the 2nd order Taylor approximation of change in relative entropy $\triangle(w, w^0) - \triangle(\hat{w}, w^0) = \sum_{ge} \frac{1}{2\hat{w}_{ge}}((w_{ge} - \hat{w}_{ge})^2 + 2\hat{w}_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}}(w_{ge} - \hat{w}_{ge})\ )$in (5.6). This is reformulated as a quadratic optimization problem and we solve it using the convex optimizer package ($http://www.stanford.edu/\sim boyd/cvx/$).

$$\boxed{\begin{aligned} min_w \quad & \sum_{ge} \frac{1}{2\hat{w}_{ge}}((w_{ge} - \hat{w}_{ge})^2 + 2\hat{w}_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}}(w_{ge} - \hat{w}_{ge})) \\ s.t. \quad & \sum_{ge} w_{ge} y_{ge} h^q(x_{ge}) \leq 0 \quad q = 1 \ldots t \\ & \sum_{ge} w_{ge} = 1 \end{aligned}} \tag{5.6}$$

5.3 Data set

We assemble a compendium of hypoxia response expression experiments in *Saccharomyces cerevisiae* that have not previously been analyzed using MEDUSA. The compendium consists of gene expression profiles of $\sim$ 5000 genes in response to 55 hypoxia related experimental conditions from two previously published microarray data sets, with three independent biological replicates for each condition. Among the 55 microarray experiments, 10 examine the temporal response of both yeast wild-type and MSN2/4 strains to short-term anaerbiosis (2 generations) in both glucose and galactose media Lai et al. (2005); 45 experiments measure the temporal response of wild-type yeast to long-term anaerbiosis (6 generations) and subsequent aerobic recovery (about 2 generations) in glucose and galactose media Lai et al. (2006). This hypoxia data is a large data set for expression profiles under hypoxia and reoxygenation in glucose versus galactose media, which is different from the small data set of perturbation experiments previously analyzed by MEDUSA Kundaje et al. (2008).

All measurements are represented as log 2 expression values, which are fold-changes with respect to a reference condition (0 time point condition for 55 time series conditions). In preprocessing, we discretize expression data by binning expression values into three states (up, down, and baseline) and partition genes into potential regulators (transcription factors and signal transducers) and targets. We download promoter sequences spanning 500 bp upstream of the genes from the Saccharomyces genome Database (SGD) and scan the promoter sequences for all occurring k-mers (k=4,5,6,7). These data are used as input to MEDUSA boosting algorithms.

5.4 Results

5.4.1 Comparison of boosting algorithms in generalization performance

5.4.1.1 Cross-validation

We perform 5-fold cross-validation experiments, randomly splitting gene-experiment examples into test and training sets with 20% of the data assigned to a test fold. We use the training set to identify statistically significant regulators and motifs and then use these regulators and motifs to predict the expression of the 20% held-out examples. We run all boosting algorithms - MEDUSA AdaBoost, MEDUSA LPBoost and MEDUSA TotalBoost for 500 boosting iterations and compare their generalization performance across 5-fold cross-validation experiments.

5.4.1.2 LPBoost vs. AdaBoost

We first compare MEDUSA AdaBoost, MEDUSA hard-margin and soft-margin LPBoost. An important parameter of the soft-margin LPBoost algorithm is the capping parameter ν. The optimal value of ν should be selected. This is accomplished by using 5-fold cross-validation on ν values in {0.01, 0.05, 0.1, 0.2, 0.3 1.0}. The best generalization performance is achieved at $\nu = 0.1$, and thus training is performed using this value for MEDUSA soft-margin LPBoost.

```
Definitions:
  $c$                         = precondition associated with a specific position in the tree
  $r_{\mu\pi\sigma}$          = weak rule associated with motif $\mu$ and regulator $\pi$ in state $\sigma$
  $w_{ge}^t$                  = weight of example (g,e) at iteration t
  $W[c(g,e)]$                 = $\sum_{c(g,e)=1} w_{ge}$, given condition c
  $\neg c$                    = not c
  $y_{ge}$                    = label of example (g,e)
  $T$                         = number of boosting iterations
  $h^t(x_{ge})$               = weak rule at iteration t
  $F^t(x_{ge})$               = prediction function at iteration t
  $Z(c,\mu,\pi,\sigma)$       = boosting loss: $W[\neg c] + 2\sqrt{W[c \wedge r_{\mu\pi\sigma}]W[c \wedge \neg r_{\mu\pi\sigma}]}$
  $\gamma_t(c,\mu,\pi,\sigma)$ = edge: $\sum_{ge} w_{ge}^t y_{ge} h^t(x_{ge})$
  $N$                         = number of training examples
  $\nu$                       = soft margin parameter
  D                           = penalization constant: $\frac{1}{N\nu}$
  $\xi_{ge}$                  = slack variables
  $\alpha_t$                  = weight of weak rule contributing to prediction score

Initialization:
  $F^0(x_{ge}) = 0$, $w_{ge}^0 = \frac{1}{N}$, for all (g,e)

Main loop:
  for $t = 1 \dots T$:
    (1) call Hierarchical Motif Clustering and get a set of proposed PSSMs
    (2) minimize boosting loss or maximize edge to obtain the new weak rule
          $h^t = argmin_{c,\mu,\pi,\sigma} Z(c,\mu,\pi,\sigma)$ or $h^t = argmax_{c,\mu,\pi,\sigma}\gamma(c,\mu,\pi,\sigma)$
    (3) optimize weights of weak rules $\alpha_q$, $q = 1 \dots t$
              $max_{\alpha,\xi,\rho}$   $\rho - D\sum_{ge}\xi_{ge}$
              s.t.          $y_{ge}\sum_{q=1}^t \alpha_q h^q(x_{ge}) + \xi_{ge} \geq \rho \quad g = 1 \dots G, e = 1 \dots E$
                            $\sum_{q=1}^t \alpha_q = 1, \xi_{ge} \geq 0$
                            $\alpha_q \geq 0, q = 1, ..., t$
    (4) update weight distribution $w_{ge}^{t+1}$:
        LPBoost: by solving dual problem of optimization problem (3):
              $min_{w^{t+1},\beta}$   $\beta$
              s.t.          $\sum_{ge} y_{ge} h^q(x_{ge}) w_{ge}^{t+1} \leq \beta \quad q = 1 \dots t$
                            $\sum_{ge} w_{ge}^{t+1} = 1,\ 0 \leq w_{ge}^{t+1} \leq D$
        TotalBoost: by minimizing the relative entropy to the initial distribution $w^0$:
                $min_w^{t+1}$   $\triangle(w^{t+1}, w^0)$
                s.t.          $\sum_{ge} w_{ge}^{t+1} y_{ge} h^q(x_{ge}) \leq 0 \quad q = 1 \dots t$
                              $\sum_{ge} w_{ge}^{t+1} = 1$
  end

Prediction of class labels:
  sign$(F^T(x_{ge})) = sign(\sum_{t=1}^T \alpha_t h^t(x_{ge}))$
```

Figure 5.1: **Pseudocode for boosting algorithm implementation.** A pseudocode description of LPBoost and TotalBoost algorithm in MEDUSA

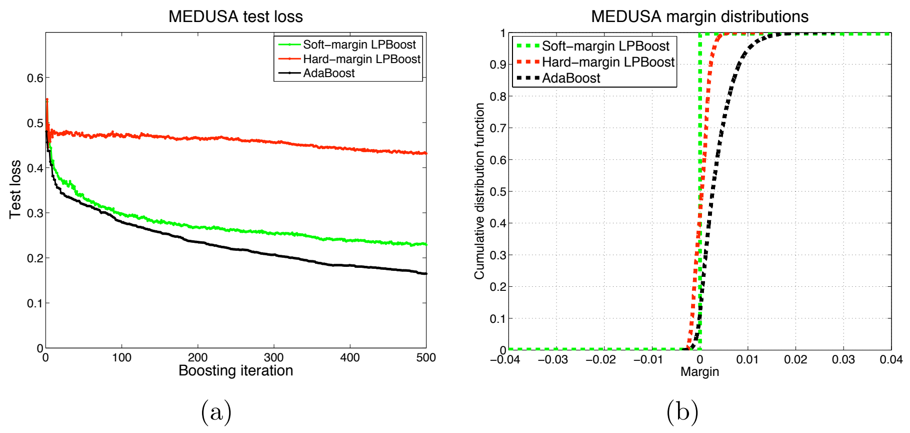


Figure 5.2: **Comparison of generalization performance of MEDUSA soft-margin LPBoost, hard-margin LPBoost and AdaBoost.** (a) Test errors for hard-margin LPBoost, soft-margin LPBoost and AdaBoost for the hypoxia dataset as a function of the number of boosting rounds. AdaBoost performs better than both soft-margin and hard-margin LPBoost on test data. (b) Cumulative *margin* distributions for hard-margin LPBoost, soft-margin LPBoost and AdaBoost after 500 boosting rounds. AdaBoost has the best *margin* distribution and hence best prediction accuracy.

Figure 5.2(a) illustrates test errors for MEDUSA AdaBoost, hard-margin LPBoost and soft-margin LPBoost based on 5-fold cross validation. After 500 boosting iterations, AdaBoost achieves the lowest test error 16.49%, while hard-margin LPBoost has 43.14% and soft-margin LPBoost 22.89%. It is clear that in terms of prediction performance, MEDUSA LPBoost does not have an advantage over MEDUSA AdaBoost.

To investigate why AdaBoost outperforms LPBoost, we perform a comparative analysis of margins for the two boosting methods. Figure 5.2(b) illustrates the cumulative distributions of margins of hard-margin LPBoost, soft-margin LPBoost and AdaBoost after 500 boosting rounds. In the Figure 5.2(b), we can see that the minimum *margin* for hard-margin LPBoost is considerably larger than AdaBoost. However, *margins* for hard-margin LPBoost are more concentrated near the lower end of the distribution, despite the fact that the minimum *margin* is higher. In fact, we estimate that 82.8% of all examples have a larger AdaBoost *margin* than corresponding LPBoost *margin*. Therefore, hard-margin LPBoost has more examples with negative *margins* than AdaBoost, explaining the much higher test error of hard-margin LPBoost. We conclude that for the hypoxia data set, although hard-margin LPBoost aims to maximize the minimum *margin*, it could sacrifice on the overall margin distribution to achieve an optimal minimum *margin*. By contrast, AdaBoost which minimizes an exponential weighting of all examples as a function of their margins, pays more attention to the entire *margin* distribution and thus performs better on the test data. This result suggests that the minimum *margin* is not as important as we expect and for linearly inseparable data like hypoxia data set.

On the other hand, soft-margin LPBoost's minimum *margin* is much smaller than that of hard-margin LPBoost and AdaBoost in the Figure 5.2(b). It is not surprising as soft-margin LPBoost introduces a penalty term allowing it to ignore noisy examples with very negative *margins*. By maximizing the soft instead of hard margin, soft-margin LPBoost seems to "shift" the attention from the minimum *margin* to the overall *margin* distribution, resulting in a better prediction accuracy than hard-margin LPBoost. However, soft-margin LPBoost's *margin* distribution is extremely concentrated near zero, as shown in the Figure 5.2(b). We find this results from a practical problem with LPBoost: classification weights w_{ge} could be very sparse, especially during the early iterations Bennett et al. (2000); Demiriz et al. (2002); Warmuth et al. (2006) of a column generation simplex algorithm. LPBoost uses the dual variables of the linear program for weights of training examples and searches for the best solutions in the dual space. Since training examples greatly outnumber weak rules, LPBoost is highly degenerate and could have multiple optimal dual solutions, corresponding to a face of the dual feasible region. Not all the optimal solutions are sparse; however, a simplex based algorithm will find the sparse extreme point solution.

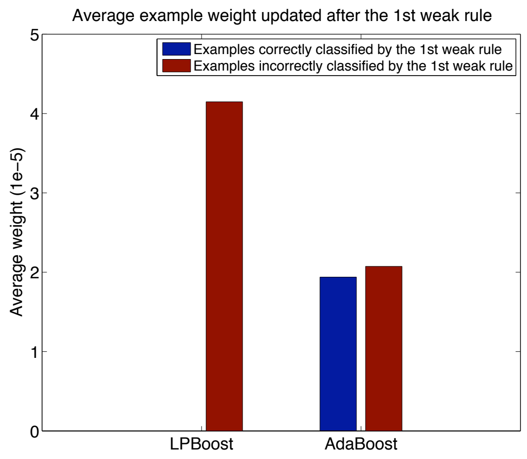


Figure 5.3: **Soft-margin LPBoost's weak rule learning overly focuses on predicting examples with negative *margin*.** We compare average weights of correctly classified and misclassified examples for soft-margin LPBoost and AdaBoost. LPBoost's weight extremely concentrates on misclassified examples, leaving weight of correctly classified ones almost zero. In contrast, for AdaBoost the average weight of the misclassified examples is only slightly higher than that of correctly classified ones.

To illustrate this point, we present a breakdown of weight of the training set after the first weak rule in Figure 5.3. We divide the training set into two groups: examples correctly classified by the first weak rule with positive *margin* and misclassified ones with negative *margin*. In Figure 5.3 we compare averaged weights of correctly classified and misclassified examples for soft-margin LPBoost and AdaBoost. We note that LPBoost's weight strongly concentrates on misclassified examples, leaving weight of correctly classified ones very close to zero. This is evidence that LPBoost produces a sparse dual solution in the optimization procedure. In contrast, for AdaBoost the average weight of misclassified examples is only slightly higher than that of correctly classified ones in Figure 5.3. Obviously AdaBoost has a much smoother weight distribution as the updated weight is an exponential function of *margins*.

Apparently, compared with AdaBoost, LPBoost overly concentrates on examples with negative *margin* and forces the learning algorithm to generate good classifications for those examples only. We estimate that among examples misclassified by the first weak rule, 67.50% are correctly classified by the second rule and 99.66% by one of the next four weak rules. This suggests that LPBoost learns highly anti-correlated weak rules that simply “flip” the example label prediction. As a result, to maximize the soft-margin LPBoost assigns almost equal scores α_i to the first few weak rules, causing the *margin* distribution to be concentrated near zero.

5.4.1.3 TotalBoost vs. AdaBoost and LPBoost

We also compare the prediction performance of MEDUSA TotalBoost vs. AdaBoost. Figure 5.4(a) plots 5-fold cross-validation test errors vs. the number of boosting iterations for soft-margin TotalBoost and AdaBoost. Similar to soft-margin LPBoost, soft-margin TotalBoost has a capping parameter ν chosen by 5-fold cross-validation. The best generalization performance is achieved at $\nu = 0.3$, and examples are trained with this optimal value. In Figure 5.4(a), we show that the test error curves of TotalBoost and AdaBoost are very close, and hence TotalBoost is competitive with AdaBoost in terms of prediction accuracy on the hypoxia data set.

Because both soft-margin LPBoost and TotalBoost are both intended to maximize the soft *margin*, we would have been very surprised to see a significant difference in their generalization error. To understand why TotalBoost outperforms LPBoost, we further examine cumulative *margin* distribution of soft-margin TotalBoost after 500 boosting iterations in Figure 5.4(b). Unlike soft-margin LPBoost, the *margin* distribution of soft-magin TotalBoost is not heavily concentrated near zero. We believe this is because TotalBoost uses entropic regularization on example weights which helps to avoid very sparse weights. To verify that TotalBoost generates a more "stable" weight distribution, we examine average weights of correctly classified and misclassified examples for soft-margin TotalBoost after the first weak rule and compare with soft-margin LPBoost and AdaBoost in Figure 5.5. As we expected, TotalBoost's average weight of the misclassified examples is only slightly higher than AdaBoost but significantly lower than LPBoost. Because of the more "uniform" weight distribution, TotalBoost does not completely "ignore" certain examples like LPBoost does and is unlikely to select new weak rules that are highly anti-correlated with old ones. TotalBoost spreads the weight to examples with higher soft margins. We believe that this tradeoff of relative entropy against pure linear optimization has a smoothing effect in the weights, which then translates into smoothed maximum *edge* and *margin* distribution. However, TotalBoost is only comparable to but not better than AdaBoost in terms of generalization error. We notice that TotalBoost has both higher minimum *margin* and average of negative *margins* than AdaBoost, yet AdaBoost has higher positive *margins*.

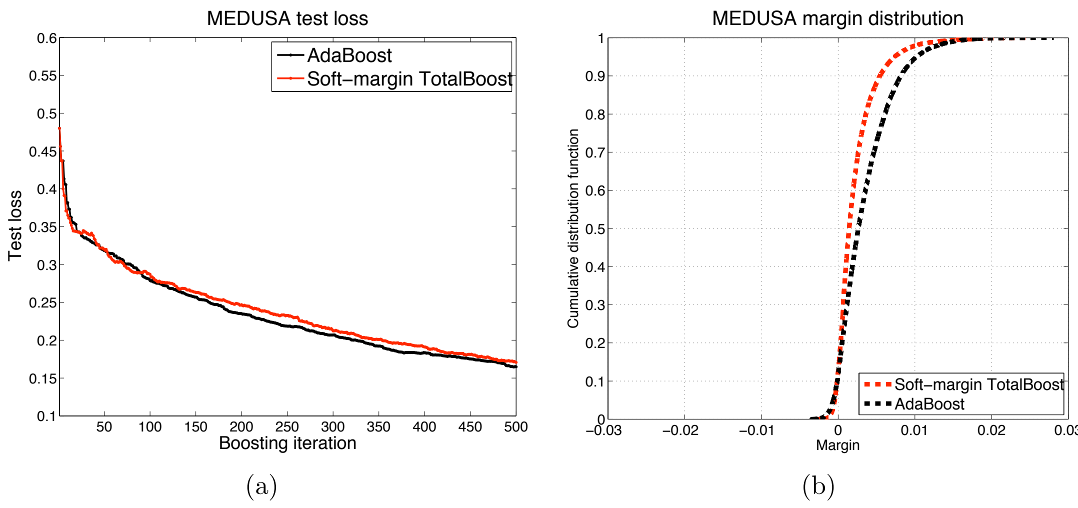


Figure 5.4: **Comparison of generalization performance of MEDUSA soft-margin TotalBoost and AdaBoost.** (a) The two boosting algorithms are run for 500 iterations and 5-fold cross-validation test errors are plotted. (b) Cumulative *margin* distributions of TotalBoost and AdaBoost.

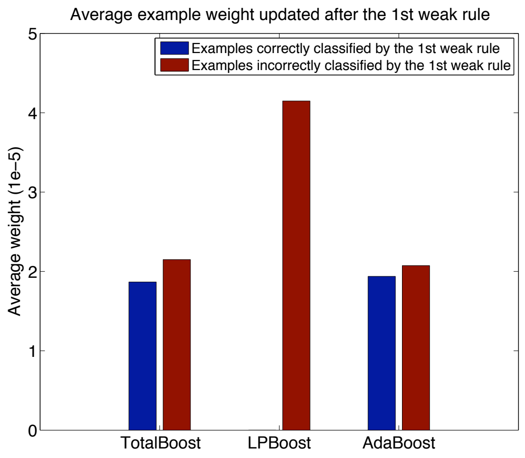


Figure 5.5: **Average weights of correctly and misclassified examples for soft-margin TotalBoost, soft-margin LPBoost and AdaBoost.** TotalBoost's average weight of the misclassified examples is only slightly higher than AdaBoost but significantly lower than LPBoost. Totalboost has a smoother weight distribution and thus does not completely "ignore" certain examples like LPBoost does.

5.4.2 MEDUSA TotalBoost retrieves a greater number of biologically meaningful motifs

MEDUSA AdaBoost and TotalBoost's high prediction accuracy give us the confidence that both boosting algorithms could retrieve statistically significant motifs. To confirm that those motifs are biologically meaningful, we compare MEDUSA AdaBoost and TotalBoost motifs against experimentally verified transcription factor (TF) binding sites compiled from two databases: TRANSFAC ($http://www.gene-regulation.com/pub/databases.html$) and SCPD ($http://rulai.cshl.edu/SCPD/$).

We measure the distance of two motifs using the average loglikelihood ratio (ALLR) calculated from `Matalign` ($http://ural.wustl.edu/software.html$). The significance of a motif match is represented by a corresponding p-value (uncorrected for multiple hypothesis testing) calculated for ALLR. We obtain p-values for all pairs of MEDUSA motifs and database binding sites. We plot the distribution of log10(p-value) in Figure 5.6 and from the distribution we set a p-value cutoff of $1.0e-5$ for a real motif match to database binding site.

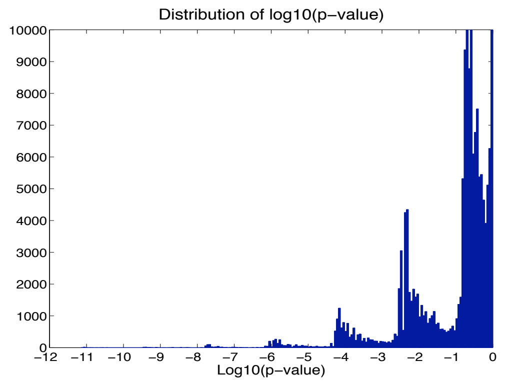
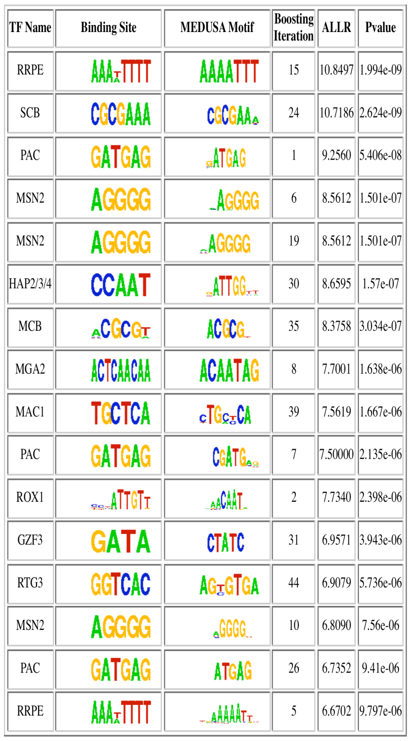


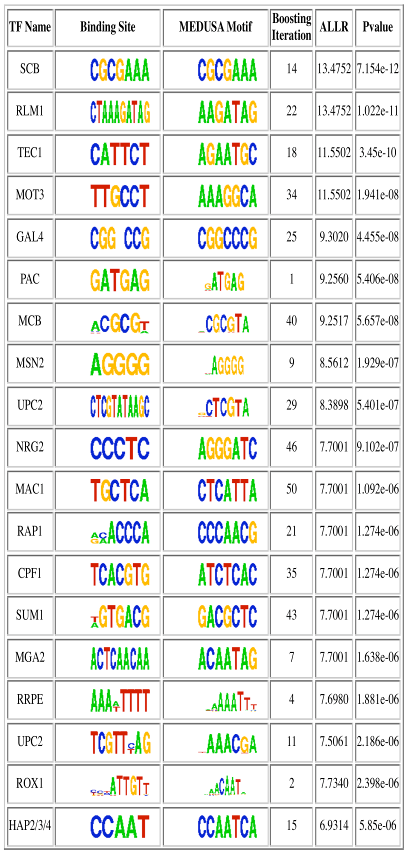
Figure 5.6: **Distribution of log10(p-value) for ALLR measuring the distance between MEDUSA motifs and database binding sites.** p-values for all pairs of MEDUSA motifs and database binding sites are estimated and a p-value cutoff of $1.0e-5$ for a real motif match to database binding site is set.

In Figure 5.7(a) and 5.7(b), we report significant motif matches (p-value $\leq 1.0e-05$), corresponding ALLRs and p-values in the order of increasing p-value for the first 50 motifs learned by MEDUSA AdaBoost and TotalBoost. If a MEDUSA motif is found to strongly match multiple known transcription factor binding site, we report the transcription factor binding site with the lowest p-value. As shown in the Figure 5.7(a) and 5.7(b), both MEDUSA AdaBoost and TotalBoost identify several important hypoxia-related transcription factor binding sites. For example, MSN2 (AGGGG) mediates stress response Segal et al. (2003b). Hap2/3/4 (CCAAT) is important for heme induction and for the regulation of oxygen-induced genes Kwast et al. (1998); Lai et al. (2005; 2006). ROX1(ATTGTT) is important for repression of hypoxic genes under aerobic conditions Kwast et al. (2002). MGA2 (ACTCAACAA) has been shown to be important for oxygen regulation of certain genes Jiang et al. (2002).



TF Name	Binding Site	MEDUSA Motif	Boosting Iteration	ALLR	Pvalue
RRPE	AAATTTT	AAAATTT	15	10.8497	1.994e-09
SCB	CGCGAAA	CGCGAA	24	10.7186	2.624e-09
PAC	GATGAG	ATGAG	1	9.2560	5.406e-08
MSN2	AGGGG	AGGGG	6	8.5612	1.501e-07
MSN2	AGGGG	AGGGG	19	8.5612	1.501e-07
HAP2/3/4	CCAAT	ATTGG	30	8.6595	1.57e-07
MCB	CGCG	ACGCG	35	8.3758	3.034e-07
MGA2	ACTCAACAA	ACAATAG	8	7.7001	1.638e-06
MAC1	TGCTCA	TGCA	39	7.5619	1.667e-06
PAC	GATGAG	CGATG	7	7.50000	2.135e-06
ROX1	ATTGT	CAAT	2	7.7340	2.398e-06
GZF3	GATA	CTATC	31	6.9571	3.943e-06
RTG3	GGTCAC	AGGTGA	44	6.9079	5.736e-06
MSN2	AGGGG	GGGG	10	6.8090	7.56e-06
PAC	GATGAG	ATGAG	26	6.7352	9.41e-06
RRPE	AAATTTT	AAAAT	5	6.6702	9.797e-06

(a)



TF Name	Binding Site	MEDUSA Motif	Boosting Iteration	ALLR	Pvalue
SCB	CGCGAAA	CGCGAAA	14	13.4752	7.154e-12
RLM1	CTAAAGATAG	AAGATAG	22	13.4752	1.022e-11
TEC1	CATTCT	AGAATGC	18	11.5502	3.45e-10
MOT3	TTGCCT	AAAGGCA	34	11.5502	1.941e-08
GAL4	CGG CCG	CGGCCCG	25	9.3020	4.455e-08
PAC	GATGAG	ATGAG	1	9.2560	5.406e-08
MCB	CGCG	CGCGTA	40	9.2517	5.657e-08
MSN2	AGGGG	AGGGG	9	8.5612	1.929e-07
UPC2	CTCGTATAAGC	CTCGTA	29	8.3898	5.401e-07
NRG2	CCCTC	AGGGATC	46	7.7001	9.102e-07
MAC1	TGCTCA	CTCATTA	50	7.7001	1.092e-06
RAP1	ACCCA	CCCAACG	21	7.7001	1.274e-06
CPF1	TCACGTG	ATCTCAC	35	7.7001	1.274e-06
SUM1	GTGACG	GACGCTC	43	7.7001	1.274e-06
MGA2	ACTCAACAA	ACAATAG	7	7.7001	1.638e-06
RRPE	AAATTTT	AAAT	4	7.6980	1.881e-06
UPC2	TCGTTAG	AAACGA	11	7.5061	2.186e-06
ROX1	ATTGT	CAAT	2	7.7340	2.398e-06
HAP2/3/4	CCAAT	CCAATCA	15	6.9314	5.85e-06

(b)

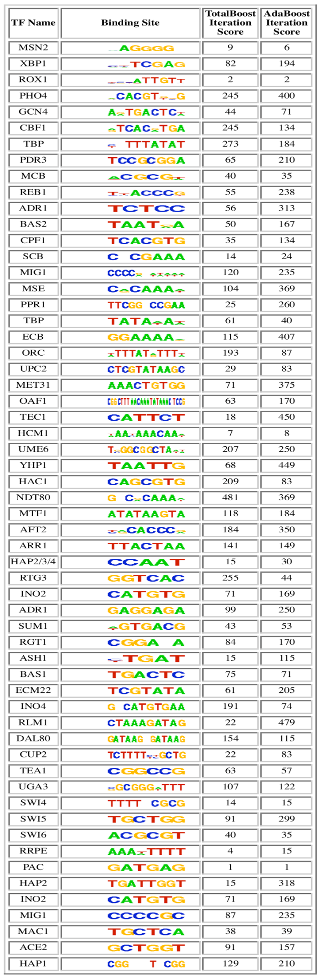
Figure 5.7: **Significant TF binding site motifs identified by MEDUSA AdaBoost and TotalBoost in the first 50 boosting iterations.** Each row in the table represents a motif found by either MEDUSA AdaBoost or TotalBoost that significantly matches a known transcription factor binding site from TRANSFAC and SCPD databases. The significance of the match is reported as a p-value for the average log likelihood ratio calculated by `Matalign` . The columns are transcription factor names, logos of transcription factor binding sites, logos of MEDUSA motifs, iteration scores (number of iteration at which MEDUSA motif is learned), average log likelihood ratios and corresponding p-values. (a) MEDUSA AdaBoost motifs that match database transcription factor binding sites (p-value $\leq 1.0e-05$). (b)MEDUSA TotalBoost motifs that match database transcription factor binding sites (p-value $\leq 1.0e-05$).

Since a large number of rRNA processing and cell cycle genes are found to be transiently down-regulated in response to anaeroboisis in galactose medium

Lai et al. (2005), both AdaBoost and TotalBoost also discover PAC (GATGAG) and RRPE (AAAWTTTT) motif elements that are significantly enriched for rRNA processing genes, and binding sites of MCB (ACGCGW) and SCB (CNCGAAA) for DNA synthesis/replication and cell cycle genes.

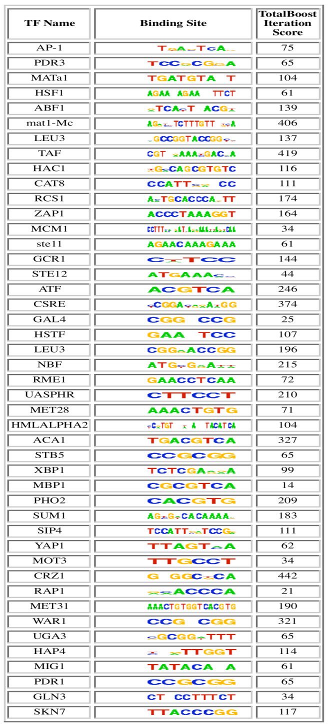
As shown in the Figure 5.7(a), AdaBoost discovers RRPE, MSN2 and PAC motifs multiple times in the first 50 iterations. This is not surprising as AdaBoost is only a "corrective" boosting algorithm and could learn motifs similar to previously selected ones. In contrast, TotalBoost, due to its "totally corrective" property, selects new weak rules that are different from earlier ones. Therefore, TotalBoost is more likely to find more unique motifs than AdaBoost after the same number of boosting iterations. As shown in the Figure 5.7(b), motifs found by both AdaBoost and TotalBoost all appear only once in the first 50 iterations of TotalBoost. In addition, TotalBoost is able to retrieve a few more transcription factor binding sites. For example, MOT3 (TTGCCT) is involved in controlling certain anaerobically expressed genes including many genes involved in cell wall maintenance and sterol biosynthesis Klinkenberg et al. (2005). UPC2 (CTCGTATA) is known to be important for oxygen and heme regulation Lai et al. (2006); Cohen et al. (2001). GAL4 (CGGNCCG) is required for the activation of the GAL genes in response to galactose Bhat and Murthy (2001).

We further perform a full comparison of MEDUSA TotalBoost to MEDUSA AdaBoost motif discovery results. We compare all 500 MEDUSA TotalBoost and AdaBoost motifs to transcription factor binding sites in TRANSFAC and SCPD. Since MEDUSA motifs learned during different boosting iterations could match to the same transcription factor binding site, we define an iteration score (IS) for the database binding site as the boosting iteration at which the matched MEDUSA motif first appears. If multiple motifs are found to be strong matches to the same transcription factor binding site, we only report the one with the lowest iteration score.



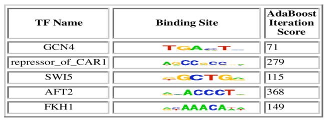
TF Name	Binding Site	TotalBoost Iteration Score	AdaBoost Iteration Score
MSN2	AGGGG	9	6
XBP1	TCGAG	82	194
ROX1	ATTGTT	2	2
PHO4	CACGT G	245	400
GCN4	A TGACTC	44	71
CBF1	TCAC TGA	245	134
TBP	TTTATAT	273	184
PDR3	TCCGCGGA	65	210
MCB	ACGCG	40	35
REB1	ACCCG	55	238
ADR1	TCTCC	56	313
BAS2	TAAT A	50	167
CPF1	TCACGTG	35	134
SCB	C CGAAA	14	24
MIG1	CCCC	120	235
MSE	C CAAA	104	369
PPR1	TTCGG CCGAA	25	260
TBP	TATA A	61	40
ECB	GGAAAA	115	407
ORC	TTTAT TTT	193	87
UPC2	CTCGTATAAGC	29	83
MET31	AAACTGTGG	71	375
OAF1	CGGCTTTAACAAATATAAAC TCCG	63	170
TEC1	CATTCT	18	450
HCM1	AA AAACAA	7	8
UME6	T GGCGGCTA	207	250
YHP1	TAATTG	68	449
HAC1	CAGCGTG	209	83
NDT80	G C CAAA	481	369
MTF1	ATATAAGTA	118	184
AFT2	CACCC	184	350
ARR1	TTACTAA	141	149
HAP2/3/4	CCAAT	15	30
RTG3	GGTCAC	255	44
INO2	CATGTG	71	169
ADR1	GAGGAGA	99	250
SUM1	GTGACG	43	53
RGT1	CGGA A	84	170
ASH1	TGAT	15	115
BAS1	TGACTC	75	71
ECM22	TCGTATA	61	205
INO4	G CATGTGAA	191	74
RLM1	CTAAAGATAG	22	479
DAL80	GATAAG GATAAG	154	115
CUP2	TCTTTT GCTG	22	83
TEA1	CGGCCG	63	57
UGA3	GCGGG TTT	107	122
SWI4	TTTT CGCG	14	15
SWI5	TGCTGG	91	299
SWI6	ACGCGT	40	35
RRPE	AAA TTTT	4	15
PAC	GATGAG	1	1
HAP2	TGATTGGT	15	318
INO2	CATGTG	71	169
MIG1	CCCCGC	87	235
MAC1	TGCTCA	38	39
ACE2	GCTGGT	91	157
HAP1	CGG T CGG	129	210

(a) Intersection of motifs discovered by AdaBoost and TotalBoost



TF Name	Binding Site	TotalBoost Iteration Score
AP-1	T A TcA	75
PDR3	TCC CG A	65
MATa1	TGATGTA T	104
HSF1	AGAA AGAA TTCT	61
ABF1	TCA T ACG	139
mat1-Mc	AG TCTTTGTT A	406
LEU3	GCCGGTACCGG	137
TAF	CGT AAA GAC A	419
HAC1	G CAGCGTGTC	116
CAT8	CCATT CC	111
RCS1	A TGCACCCA TT	174
ZAP1	ACCCTAAAGGT	164
MCM1	CCTTT AT A AA CAA	34
ste11	AGAACAAAGAAA	61
GCR1	C TCC	144
STE12	ATGAAA	44
ATF	ACGTCA	246
CSRE	CGGA A GG	374
GAL4	CGG CCG	25
HSTF	GAA TCC	107
LEU3	CGG ACCGG	196
NBF	ATG G A	215
RME1	GAACCTCAA	72
UASPHR	CTTCCT	210
MET28	AAACTGTG	71
HMLALPHA2	C TGT A TACATCA	104
ACA1	TGACGTCA	327
STB5	CCGCGG	65
XBP1	TCTCGA A	99
MBP1	CGCGTCA	14
PHO2	CACGTG	209
SUM1	AG G CACAAAA	183
SIP4	TCCATT TCCG	111
YAP1	TTAGT A	62
MOT3	TTGCCT	34
CRZ1	G GGC CA	442
RAP1	ACCCA	21
MET31	AAACTGTGGTCACGTG	190
WAR1	CCG CGG	321
UGA3	GCGG TTT	65
HAP4	TTGGT	114
MIG1	TATACA A	61
PDR1	CCGCGG	65
GLN3	CT CCTTTCT	34
SKN7	TTACCCGG	117

(b) Motifs discovered by TotalBoost but not AdaBoost



TF Name	Binding Site	AdaBoost Iteration Score
GCN4	TGA T	71
repressor_of_CAR1	A CC CC	279
SWI5	GCTG	115
AFT2	ACCCT	368
FKH1	AAACA	149

(c) Motifs discovered by AdaBoost but not TotalBoost

Figure 5.8: **Comparison of TF binding site motifs learned by MEDUSA AdaBoost and TotalBoost.** Each row in the table represents a motif found by both MEDUSA AdaBoost and TotalBoost (left section), by MEDUSA TotalBoost only (middle section), or by MEDUSA AdaBoost only (right section). The first column describes the motif by the name of the corresponding transcription factor, second column logo of TF binding site motif, third column the iteration score of the motif. In the left table, two iteration scores are reported and compared for both AdaBoost and TotalBoost.

We show in Figure 5.8 unique MEDUSA motifs that match known transcription factor binding sites and corresponding iteration scores. In total, we match 108 motifs found by either or both boosting algorithms to known binding sites and we sort those motifs into 3 categories. Figure 5.8(a) shows the first set consisting of 58 motifs identified by both MEDUSA TotalBoost and AdaBoost. Figure 5.8(b) demonstrates the second set consisting of 45 motifs identified by MEDUSA TotalBoost but not MEDUSA TotalBoost. Figure 5.8(c) shows the third set consisting of 5 motifs found by MEDUSA AdaBoost but not MEDUSA AdaBoost. In all the three figures, iteration scores of those unique MEDUSA motifs matching to transcription factor binding sites are also reported. Both MEDUSA TotalBoost and Adaboost identify a number of transcription factor binding sites, including ones that regulate genes involved in G1/S transition of the cell cycle (MCB, SCB, SWI4, SWI6), ribosomal biogenesis (PAC, PPRE), respiration (HAP2/3/4), oxidative stress response (MSN2/4, AFT2), carbohydrate usage and anaerobic redox balance (ROX1), sterol and cell wall maintenance (UPC2), amino acid biosynthesis and starvation response (GCN4), glycerol metabolism and osmolarity (ADR1), glucose metabolism and carbon source based stress (MIG1), carbohydrate metabolism (ACE2), methionine biosynthesis (MET31), mitochondrial (RTG1) and purine biosynthesis (BAS1). However, MEDUSA TotalBoost is able to find 45 more transcription factor binding sites, as shown in the Figure 5.8(b). These transcription factors include ones that control genes involved in stress response (HSF1, XBP1), ribosomal biogenesis (ABF1), cell cycle (STE12), galactose metabolism (GAL4), meiosis (UME1), sulphur metabolism (MET28), YAP1 (stress response), sterol and cell wall maintenance (MOT3). In contrast, there are only 5 binding sites identified by MEDUSA AdaBoost only. These results show that MEDUSA TotalBoost outperforms MEDUSA AdaBoost in finding a greater number of biologically meaningful motifs for our hypoxia dataset. In addition, we find that among the the 58 motifs found by both MEDUSA TotalBoost and AdaBoost in the Figure 5.8(a), 44 have lower iteration scores in MEDUSA TotalBoost than MEDUSA AdaBoost. In Figure 5.9, we plot iterations scores of the 58 motifs in AdaBoost versus TotalBoost, and show that significantly more motifs have lower iteration scores in TotalBoost than AdaBoost. This is evidence that motifs learned by both boosting algorithms tend to appear earlier in TotalBoost than AdaBoost. Since in our MEDUSA model motifs selected at early boosting iterations are more important than ones selected later, this finding suggests that TotalBoost is more efficient in extracting significant biological features than AdaBoost does.

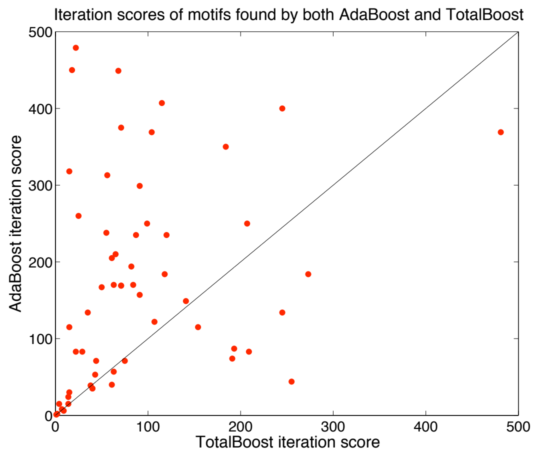


Figure 5.9: **TotalBoost vs. AdaBoost: iteration scores of motifs.** Every motif among the 58 found by both MEDUSA TotalBoost and AdaBoost is assigned both an AdaBoost and TotalBoost iteration score. We compare the iteration scores and find that 44 motifs have lower scores in TotalBoost, suggesting that TotalBoost retrieves more motifs at earlier boosting rounds.

MEDUSA TotalBoost identifies motifs relevant for specific gene sets

MEDUSA TotalBoost identifies more motifs matching transcription factor binding sites than AdaBoost. We further analyze whether MEDUSA TotalBoost could correctly extract regulatory information about a specific gene set. In particular, we are interested in identifying the most significant MEDUSA motifs controlling specific sets of target genes under specific environmental conditions. To do that, we rank motifs using a margin-based score, which assesses how significantly the motif contributes to the confidence of prediction over a gene set in specific experiments.

If we remove, for example, motif m from the regulatory program (i.e. delete nodes containing m and their subtrees from the ADT learned by boosting), we denote F^{-m} as the modified prediction function and define the following margin-based score:

$$S_m = \frac{1}{|N|} \sum_{\{(x_{ge}, y_{ge})\} \in N} y_{ge}(F(x_{ge}) - F^{-m}(x_{ge})).$$

Here N is the specific set of training examples considered. The score S_m will again be positive if on average m is important for making predictions, and its size measures its importance to the target set.

The hypoxia dataset consists of gene expression profiles in 55 experiments. 45 experiments examine the temporal response to long-term anaerobiosis and subsequent aerobic recovery in galactose and glucose media Lai et al. (2006). Previous studies have revealed a group of oxygen-responsive genes whose expression levels respond significantly to shifts in oxygen availability in galactose medium Lai et al. (2006). This gene group is partitioned into 18 gene signatures through clustering analysis of gene expression profiles in Lai et al. (2006). The 18 gene signatures have diverse temporal profiles, and we use margin scoring to identify signficant motifs for those gene signatures.

We first examine gene signatures 9, 10 and 11. The 9th gene signature identified in galactose medium contains genes that are chronically down-regulated during anaerobiosis yet rapidly return to normoxic levels upon reoxygenation. This temporal signature suggests positive regulation by heme Kwast et al. (1998); Lai et al. (2005; 2006). This group of oxygen-responsive genes are mostly involved in mitochondrial functions such as respiration and energy metabolism and are largely controlled by heme-responsive factors HAP1 and HAP2/3/4 Lai et al. (2006). Since this gene group exhibits a similar expression pattern in glucose medium, we perform margin score analysis across all experiments in both galactose and glucose media. We find that MEDUSA TotalBoost ranks motifs that match to binding sites of HAP1 (CGGNNTANCGG) and HAP2/3/4 (CCAAT) as important features for this heme-regulated gene group. The 10th and 11th gene signatures in galactose medium contain the majority of genes that are also chronically down-regulated during anaerobosis but rapidly up-regulated upon reoxygenation. These two gene sets are not only involved in mitochondrial functions like respiration, but also cellular processes including sterol synthesis, lipid metabolism and oxidative stress response Lai et al. (2006). Similarly, we identify binding sites of HAP1 and MSN2/4 as high-scoring motifs for both gene signatures in galactose medium, and also SKN7 for the 11th gene signature in galactose medium. MSN2/4 plays a key role in stress response Gasch et al. (2000) and SKN7 is a known transcription factor regulating oxidative stress response Morgan et al. (1997). As shown in table 5.8(a), MEDUSA AdaBoost discovers binding sites for HAP1, HAP2/3/4 and MSN2/4. However, MEDUSA AdaBoost is not able to identify the binding site for SKN7, a more subtle context-specific transcription factor regulating a subset of the 11th gene signature involved in oxidative stress response.

The 13th gene signature contains genes that are transiently up-regulated in response to anaerobiosis Lai et al. (2006). Previous analysis revealed that this up-regulation response to anaerobiosis was similar to “environmental stress response” Gasch et al. (2000). As many up-regulated genes are not found to be differentially expressed during O_2 shifts in glucose medium, the “stress” is believed to be associated with the acute cessation of respiration during the metabolic transition rather than O_2 deprivation. Since the temporal expression profile of this gene signature is only unique to galactose medium, we perform margin score analysis on this gene signature across galactose medium experiments only. We identify MSN2/4 binding site (AGGGG) is as the top-ranked motif. This is not surprising because MSN2 and MSN4 are known to be important transcription factors regulating the response to environmental stress conditions Gasch et al. (2000). A motif found by MEDUSA TotalBoost but not MEDUSA AdaBoost is the binding site for transcription factor HSF1, which also has been shown to be important for stress response particularly heat shock response Morano et al. (1999); Grably et al. (2002). Here we find this motif have a high margin score. In addition, we find support for HSF1 regulating at least several of the target genes in this gene signature. For example, HSP genes are known to be regulated by HSF1 and the HSF1 binding site is present in HSP42, HSP70, HSP78 and HSP104 from this gene signature Yamamoto et al. (2005).

The 3rd and 4th gene signatures contain the majority of genes that are acutely yet transiently down-regulated in response to anaerobiosis. Most genes in the 3rd and 4th gene signatures are found to be involved in early steps of cytoplasmic ribosomal biogenesis, particularly rRNA and tRNA synthesis/processing Lai et al. (2006). The transient repression of those genes involved in rRNA and tRNA processing/transcription is associated with reducing the energy demand, in response to the abrupt cessation of galactose-supported respiration Lai et al. (2006). MEDUSA TotalBoost and AdaBoost identify PAC and RRPE elements, both of which are known to mediate the expression of enzymes and proteins involved in ribosome biogenesis Taddei et al. (2009); Lai et al. (2005). And our margin score analysis reveal these two binding sites as high-scoring motifs for both gene signatures. In addition, MEDUSA TotalBoost finds a motif similar to the binding site of ABF1, which regulates genes involved in a wide range of cellular processes including ribosomal functions Della Seta et al. (1990); Planta et al. (1995). However, unlike PAC and RRPE, ABF1 motif only has a low positive margin score for the 4th gene signature.

The genes in the 15th and 16th signatures exhibit chronic anaerobic up-regulation and return to pre-anoxic levels after reoxygenation. This expression pattern suggests negative regulation by heme Kwast et al. (1998). The binding site of ROX1, a known heme-responsive transcription factor, is identified by MEDUSA TotalBoost as a top motif by margin score for both signatures across galactose experiments. Moreover, the function of many genes in these two signatures are consistent with the role of ROX1, particularly in regulating carbohydrate catabolism and redox balance Kwast et al. (2002); Lai et al. (2006). The 17th and 18th gene signatures also contain genes that are chronically up-regulated under anaerobiosis but only after a delay ($\geq$ 2 generations) Lai et al. (2006). Similarly, the ROX1 motif also has a high margin score for signature 17 and 18. In addition, MEDUSA TotalBoost learns binding site of UPC2, which is negatively regulated by ROX1 in heme-regulated networks Kwast et al. (1998). Previous studies have shown that many genes in signature 17 and 18 contain UPC2 binding sites in their promoters Lai et al. (2006), so it is not surprising that UPC2 motif also ranks high by margin score in MEDUSA TotalBoost. Finally, MEDUSA TotalBoost finds another heme-dependment trancription factor MOT3, which is not discovered by MEDUSA AdaBoost. Compared with ROX1 and UPC2, MOT3 motif has a relatively low margin score. MOT3 seems to be important for only a subset of target genes in the gene signature involved in cell wall maintenance and sterol biosynthesis Klinkenberg et al. (2005); Lai et al. (2006).

5.5 Conclusion

We compare three boosting algorithms - AdaBoost, LPBoost and TotalBoost in the MEDUSA model to predict gene expression and learn motifs for hypoxia response in Saccharomyces cerevisiae. A comparative analysis of generalization performance of AdaBoost, LPBoost and TotalBoost algorithms has been performed.

Results from the analysis reveal that TotalBoost has generalization error comparable to AdaBoost, and they both outperform LPBoost on the hypoxia data. The underperformance of both hard-marigin and soft-margin LPBoost algorithms are counter-intuitive at first as it seems to contradict what is predicted by large-margin theory. We explain the results by considering the *margin* and weight distribution of training examples.

On the other hand, soft-margin TotalBoost has prediction performance comparable to AdaBoost. TotalBoost and AdaBoost's high prediction accuracy on the test data confirm that both boosting algorithms retrieve statistically significant binding site motifs. A comprehensive comparison with AdaBoost demonstrates that TotalBoost is more successful in identifying biologically meaningful binding site motifs. First, TotalBoost identifies more "unique" motifs than AdaBoost during early boosting rounds. We believe TotalBoost outperforms AdaBoost in this aspect because of its *totally corrective* property, meaning that TotalBoost attempts to select new weak rules that are "decorrelated" with all past ones. Second, the total number of motifs found by TotalBoost that match known transcription factor binding sites in the database is larger than that of AdaBoost. For motifs found by both TotalBoost and AdaBoost, TotalBoost tends to learn them earlier than AdaBoost, making TotalBoost a more efficient boosting algorithm for extracting biological features in the MEDUSA model.

Finally, we study if MEDUSA TotalBoost can be used to explain the context-specific regulation of target genes in terms of the presence of binding motifs that mediate regulatory control. To do this, we use a margin-based score to identify the most significant motifs for specific sets of target genes under specific conditions. We show that TotalBoost could retrieve motifs, including ones not identified by AdaBoost, that have high margin scores for relevant genes sets and experimental conditions. In summary, our results suggest that MEDUSA TotalBoost achieves better performance than AdaBoost in motif discovery and could offer new biological insights.

Chapter 6

Conclusions and Outlook

Computational biophysics is a fast growing field. Studying gene regulation has been one of the main challenges in computational biophysics and it has many applications in medicine, e.g. cancer diagnostics. With large amounts of data available, scientists now can systematically model and predict gene regulation in many organisms. In this thesis, we have presented three case studies that utilize machine learning algorithms to study gene regulatory networks, particularly the protein-DNA interactions of transcription factors and the short sequence elements to which they bind to regulate gene expression.

Many computational models have focused on structure learning from the gene expression data. For example, clustering genes to discover groups of potentially co-regulated genes or constructing a graph with potential regulatory "edges" between genes. In contrast, we are interested in predictive modeling of gene regulation. Our models learn biological features that help to predict expression profiles of genes. Since the models are based on supervised machine learning algorithms, we can show that learned features are statistically significant, which is associated with high generalization instead of simple correlations in the data.

This thesis demonstrates how learning algorithms can be applied to the biophysics problems. Computational biophysics is a very interdisciplinary field. Scientists from different fields including physics, mathematics, biology and computer science have collaborated to work on the computational projects. Physicists can bring rigorous mathematical thinking to this field, since the mathematics involved in physical theories is generally more complicated than other sciences. With a strong background in mathematical modeling, physicists can easily understand and interpret underlying patterns in data, build new learning models with clearly defined assumptions in mathematical formulas and define quantitative metrics that evaluate model performance. We believe physicists can make contributions to computational biophysics by modeling complex systems and proposing biological hypotheses.

We hope our work will motivate more physicists to work in biophysics. In addition, we believe many fields other than biophysics can benefit from the application of machine learning tools. More specifically, learning methods help to analyze experimental data, guide experiment design and model building, improve our understanding of complex systems and answer important questions of interest to many communities.

Bibliography

Abazov, V., B. Abbott, A. Abdesselam, M. Abolins, V. Abramov, B. Acharya, D. Adams, M. Adams, S. Ahmed, G. Alexeev, et al. (2001). Search for single top quark production at dø using neural networks. *Physics Letters B 517*(3-4), 282–294.

Ashburner, M., C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, et al. (2000). Gene ontology: tool for the unification of biology. *Nature genetics 25*(1), 25.

Bader, G. D. and C. W. V. Hogue (2003). An automated method for finding molecular complexes in large protein interaction networks. *BMC Bioinformatics 4*, 2.

Bailey, T. L. and C. Elkan (1994a). Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pp. 28–36. AAAI Press.

Bailey, T. L. and C. Elkan (1994b). Fitting a mixture model by expectation maximization to discover motifs in biopolymers. *Proc Int Conf Intell Syst Mol Biol. 2*, 28–36.

Baugh, L. R., A. A. Hill, D. K. Slonim, E. L. Brown, and C. P. Hunter (2003). Composition and dynamics of the Caenorhabditis elegans early embryonic transcriptome. *Development 130*(5), 889–900.

Beer, M. A. and S. Tavazoie (2004, Apr 16). Predicting gene expression from sequence. *Cell 117*(2), 185–98.

Bennett, K., A. Demiriz, and J. Shawe-Taylor (2000). A column generation algorithm for boosting. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 65–72. Citeseer.

Bergmann, S., J. Ihmels, and N. Barkai (2003, Mar). Iterative signature algorithm for the analysis of large-scale gene expression data. *Phys Rev E Stat Nonlin Soft Matter Phys. 67(3 Pt 1)*, 031902. Epub 2003 Mar 11.

Berman, B., Y. Nibu, B. Pfeiffer, P. Tomancak, S. Celniker, M. Levine, G. Rubin, and M. Eisen (2002). Exploiting transcription factor binding site clustering to identify cis-regulatory modules involved in pattern formation in the drosophila genome. *Proceedings of the National Academy of Sciences 99*, 757–762.

Bhat, P. and T. Murthy (2001). Transcriptional control of the GAL/MEL regulon of yeast Saccharomyces cerevisiae: mechanism of galactose-mediated signal transduction. *Molecular Microbiology 40*(5), 1059–1066.

Bigelow, H. R., A. S. Wenick, A. Wong, and O. Hobert (2004, Mar 12). CisOrtho: a program pipeline for genome-wide identification of transcription factor target genes using phylogenetic footprinting. *BMC Bioinformatics. 5*, 27.

Bonneau, R., D. Reiss, P. Shannon, M. Facciotti, L. Hood, N. Baliga, and V. Thorsson (2006). The inferelator: an algorithm for learning parsimonious regulatory networks from systems-biology data sets de novo. *Genome Biol 7*(5), R36.

Boulesteix, A.-L. and K. Strimmer (2007). Partial least squares: a versatile tool for the analysis of high-dimensional genomic data. *Brief Bioinform 8*(1), 32–44.

Breiman, L. (1998). Arcing classifiers. *The annals of statistics 26*(3), 801–824.

Brilli, M., R. Fani, , and P. Lió (2006). MotifScorer: using a compendium of microarrays to identify regulatory motifs. *Bioinformatics 23*, 493–495.

Bussemaker, H. J., H. Li, and E. D. Siggia (2001). Regulatory element detection using correlation with expression. *Nat Genet 27*, 167–171.

Chung, F. R. K. (1997, February). *Spectral Graph Theory (CBMS Regional Conference Series in Mathematics, No. 92) (Cbms Regional Conference Series in Mathematics).* American Mathematical Society.

Cohen, B., O. Sertil, N. Abramova, K. Davies, and C. Lowry (2001). Induction and repression of DAN1 and the family of anaerobic mannoprotein genes in Saccharomyces cerevisiae occurs through a complex array of regulatory sites. *Nucleic Acids Research 29*(3), 799.

Costanzo, M. C., J. D. Hogan, M. E. Cusick, B. P. Davis, A. M. Fancher, P. E. Hodges, P. Kondu, C. Lengieza, J. E. Lew-Smith, C. Lingner, K. J. Roberg-Perez, M. Tillberg, J. E. Brooks, and J. I. Garrels (2000). The yeast proteome database (YPD) and Caenorhabditis elegans proteome database (WormPD): comprehensive resources for the organization and comparison of model organism protein information. *Nucleic Acids Res 28*(1), 73–76.

CR, R. (1993). *Linear Statistical Inference and its Application.* Wiley.

Culetto, E., D. Combes, Y. Fedon, A. Roig, J. P. Toutant, and M. Arpagaus (1999, Jul 30). Structure and promoter activity of the 5' flanking region of ace-1, the gene encoding acetylcholinesterase of class A in Caenorhabditis elegans. *J Mol Biol. 290*(5), 951–66.

Della Seta, F., S. Ciafre, C. Marck, B. Santoro, C. Presutti, A. Sentenac, and I. Bozzoni (1990). The ABF1 factor is the transcriptional activator of the L2 ribosomal protein genes in Saccharomyces cerevisiae. *Molecular and cellular biology 10*(5), 2437.

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

D'Haeseleer, P., X. Wen, S. Fuhrman, and R. Somogyi (1999). Linear modeling of mRNA expression levels during CNS development and injury. *Pacific Symposium on Biocomputing*.

Drucker, H. and C. Cortes (1996). Boosting decision trees. *Advances in neural information processing systems*, 479–485.

Eskin, E., M. Gelfand, and P. Pevzner (2002). Genome wide analysis of bacterial promoter regions. In *Biocomputing 2003: Proceedings of the Pacific Symposium Hawaii, USA 3-7 January 2003*, pp. 29. World Scientific Pub Co Inc.

et al., T. S. S. (1998). Comprehensive identification of cell cycle-related genes of the yeast Saccharomyces cerevisiae by microarray hybridization. *Mol. Biol. of the Cell 9*, 3273–3297.

Freund, Y. and L. Mason (1999a). The alternating decision tree learning algorithm. *Proceedings of the Sixteenth International Conference on Machine Learning*, 124–133.

Freund, Y. and L. Mason (1999b). The alternating decision tree learning algorithm. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 124–133. Citeseer.

Freund, Y. and R. Schapire (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. In *Computational learning theory*, pp. 23–37. Springer.

Freund, Y. and R. Schapire (1996). Experiments with a new boosting algorithm. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pp. 148–156. Citeseer.

Freund, Y., R. Schapire, and N. Abe (1999). A short introduction to boosting. *JOURNAL-JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE 14*, 771–780.

Freund, Y. and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences 55*(1), 119–139.

Friedman, J., R. Olshen, C. Stone, A. S. Association, and A. S. A. A. Meeting (1986). Classification and regression trees. American Statistical Association.

Friedman, N. (2004). Inferring cellular networks using probabilistic graphical models. *Science 303*, 799–805.

Gasch, A. P., P. T. Spellman, C. M. Kao, O. Carmel-Harel, M. B. Eisen, G. Storz, D. Botstein, and P. O. Brown (2000). Genomic expression programs in the response of yeast cells to environmental changes. *Mol. Biol. Cell 11*, 4241–4257.

Geladi, P. and B. Kowalski (1986). Partial least-squares regression: a tutorial. *Analytica Chimica Acta 185*, 1–17.

Grably, M., A. Stanhill, O. Tell, and D. Engelberg (2002). HSF and Msn2/4p can exclusively or cooperatively activate the yeast HSP104 gene. *Molecular microbiology 44*(1), 21–35.

Grove, A. and D. Schuurmans (1998). Boosting in the limit: Maximizing the margin of learned ensembles. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 692–699. JOHN WILEY & SONS LTD.

Grumbling, G. and V. Strelets (2006, Jan 1). FlyBase: anatomical data, images and queries. *Nucleic Acids Res. 34(Database issue)*, D484–8.

Hartemink, A. J., D. K. Gifford, T. S. Jaakkola, and R. A. Young (2001). Using graphical models and genomic expression data to statistically validate models of genetic regulatory networks. *Pacific Symposium on Biocomputing*.

Hastie, T., R. Tibshirani, and J. Friedman (2001). *The Elements of Statistical Learning*. Springer-Verlag, NY, USA.

Hertz, G. Z. and G. D. Stormo (1999, Jul-Aug). Identifying and protein patterns with statistically significant alignments of multiple sequences. *Bioinformatics 15(7-8)*, 563–77.

Hoffmann, R., V. Minkin, and B. Carpenter (1997). Ockhams razor and Chemistry. *HYLE Int. J. Phil. Chem 3*(3), 28.

Hopfield, J. (1982). Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America 79*(8), 2554.

Hughes, J., P. Estep, S. Tavazoie, and G. Church (2000a). Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *J Mol Biol 296*(5), 1205–1214.

Hughes, J. D., P. W. Estep, S. Tavazoie, and G. M. Church (2000b). Computational identification of cis-regulatory elements associated with groups of functionally related genes in Saccharomyces cerevisiae. *Journal of Molecular Biology 296*(5), 1205–14.

J., C. and S. M. K. (1999). Basic helix-loop-helix proteins can act at the e-box within the serum response element of the c-fos promoter to influence hormone-induced promoter activation in sertoli cells. *Mol Endocrinol 13*(5), 774–86.

Janssens, H., S. Hou, J. Jaeger, A. R. Kim, E. Myasnikova, D. Sharp, and J. Reinitz. Quantitative and predictive model of transcriptional control of the Drosophila melanogaster even skipped gene.

Jiang, Y., M. Vasconcelles, S. Wretzel, A. Light, L. Gilooly, K. McDaid, C. Oh, C. Martin, and M. Goldberg (2002). Mga2p processing by hypoxia and unsaturated fatty acids in Saccharomyces cerevisiae: impact on LORE-dependent gene expression. *Eukaryotic Cell 1*(3), 481.

Jong, S. (1993). SIMPLS: An alternative approach to partial least squares regression. *Chemom. Intell. Lab. Syst. 18*, 251–263.

Kearns, M. and L. Valiant (1988). *Learning Boolean formulae or finite automata is as hard as factoring.* Harvard University, Center for Research in Computing Technology, Aiken Computation Laboratory.

Kivinen, J. and M. Warmuth (1999). Boosting as entropy projection. In *Proceedings of the twelfth annual conference on Computational learning theory*, pp. 134–144. ACM New York, NY, USA.

Klinkenberg, L., T. Mennella, K. Luetkenhaus, and R. Zitomer (2005). Combinatorial repression of the hypoxic genes of Saccharomyces cerevisiae by DNA binding proteins Rox1 and Mot3. *Eukaryotic cell 4*(4), 649.

Kundaje, A., S. LIANOGLOU, X. LI, D. QUIGLEY, M. Arias, C. Wiggins, L. Zhang, and C. Leslie (2007). Learning regulatory programs that accurately predict differential expression with medusa. *Annals of the New York Academy of Sciences 1115*(1), 178–202.

Kundaje, A., X. Xin, C. Lan, S. Lianoglou, M. Zhou, L. Zhang, and C. Leslie (2008). A predictive model of the oxygen and heme regulatory network in yeast. *PLoS computational biology 4*(11), e1000224.

Kwast, K., P. Burke, and R. Poyton (1998). Oxygen sensing and the transcriptional regulation of oxygen-responsive genes in yeast. *The Journal of experimental biology 201*(Pt 8), 1177.

Kwast, K., L. Lai, N. Menda, D. James III, S. Aref, and P. Burke (2002). Genomic analyses of anaerobically induced genes in Saccharomyces cerevisiae: functional roles of Rox1 and other factors in mediating the anoxic response. *Journal of bacteriology 184*(1), 250.

Lai, L., A. Kosorukoff, P. Burke, and K. Kwast (2005). Dynamical Remodeling of the Transcriptome during Short-Term Anaerobiosis in Saccharomyces cerevisiae: Differential Response and Role of Msn2 and/or Msn4 and Other Factors in Galactose and Glucose Media Supplemental material for this article may be found at http://mcb. asm. org/. *Molecular and Cellular Biology 25*(10), 4075–4091.

Lai, L., A. Kosorukoff, P. Burke, and K. Kwast (2006). Metabolic-state-dependent remodeling of the transcriptome in response to anoxia and subsequent reoxygenation in Saccharomyces cerevisiae. *Eukaryotic Cell 5*(9), 1468.

Lawrence, C. E., S. F. Altschul, M. S. Boguski, J. S. Liu, A. F. Neuwald, and J. C. Wootton (1993). Detecting subtle sequence signals: A Gibbs sampling strategy for multiple alignment. *Science 262*(5131), 208–214.

Li, X., C. Panea, C. Wiggins, V. Reinke, and C. Leslie (2010). Learning graph-mer motifs that predict gene expression trajectories in development. *PLoS Comput Biol 6*(4), e1000761.

Lipshutz, R., S. Fodor, T. Gingeras, D. Lockhart, et al. (1999). High density synthetic oligonucleotide arrays. *Nature genetics 21*(Suppl 1), 20–24.

Luenberger, D. and Y. Ye (2008). *Linear and nonlinear programming.* Springer Verlag.

Martens, H. (2001). Reliable and relevant modelling of real world data: a personal account of the development of PLS regression. *Chemometrics and intelligent laboratory systems 58*(2), 85–95.

Martens, H. and T. Naes (1992). *Multivariate calibration.* John Wiley & Sons Inc.

Matys, V., O. V. Kel-Margoulis, E. Fricke, I. Liebich, S. Land, A. Barre-Dirrie, I. Reuter, D. Chekmenev, M. Krull, K. Hornischer, N. Voss, P. Stegmaier, B. Lewicki-Potapov, H. Saxel, A. E. Kel, and E. Wingender (2006). TRANSFAC and its module TRANSCompel: transcriptional gene regulation in eukaryotes. *Nucleic Acids Res 34*(Database issue), D108–110.

Middendorf, M., A. Kundaje, M. Shah, Y. Freund, C. Wiggins, and Leslie.C. (2005). Motif discovery through predictive modeling of gene regulation. In e. a. Miyano S, Mesirov J (Ed.), *RECOMB*, pp. 538–552. Cambridge, MA: Springer.

Middendorf, M., A. Kundaje, C. Wiggins, Y. Freund, and C. Leslie (2004). Predicting genetic regulatory response using classification. *Proceedings of the Twelfth International Conference on Intelligent Systems for Molecular Biology (ISMB 2004).*

Morano, K., N. Santoro, K. Koch, and D. Thiele (1999). A trans-activation domain in yeast heat shock transcription factor is essential for cell cycle progression during stress. *Molecular and cellular biology 19*(1), 402.

Morgan, B., G. Banks, W. Toone, D. Raitt, S. Kuge, and L. Johnston (1997). The Skn7 response regulator controls gene expression in the oxidative stress response of the budding yeast Saccharomyces cerevisiae. *The EMBO Journal 16*(5), 1035–1044.

Mukherjee, I., C. Rudin, and R. Schapire (2011). The rate of convergence of AdaBoost. In *Proceedings of the 24th Annual Conference on Learning Theory (COLT).*

Natarajan, L., B. M. Jackson, E. Szyleyko, and D. M. Eisenmann (2004, Aug 15). Identification of evolutionarily conserved promoter elements and amino acids required for function of the C. elegans beta-catenin homolog BAR-1. *Dev Biol. 272*(2), 536–57.

Naughton, B. T., E. Fratkin, S. Batzoglou, and D. L. Brutlag (2006). A graph-based motif detection algorithm models complex nucleotide dependencies in transcription factor binding sites. *Nucleic Acids Res 34*(20), 5730–5739.

Nguyen, D. and D. Rocke (2002). Tumor classification by partial least squares using microarray gene expression data. *Bioinformatics 18*(1), 39.

Nguyen, D. and D. Rocke (2004). On partial least squares dimension reduction for microarray-based classification: a simulation study. *Computational statistics & data analysis 46*(3), 407–425.

Nocedal, J. and S. Wright (1999). *Numerical optimization.* Springer.

Pe'er, D., A. Regev, G. Elidan, and N. Friedman (2001). Inferring subnetworks from perturbed expression profiles. *Proceedings of the Ninth International Conference on Intelligent Systems for Molecular Biology*.

Pe'er, D., V. Regev, and A. Tanay (2002). A fast and robust method to infer and characterize and active regulator set for molecular pathways. *Proceedings of the Tenth International Conference on Intelligent Systems for Molecular Biology*.

Perkins, T. J., J. Jaeger, J. Reinitz, and L. Glass (2006, May). Reverse engineering the gap gene network of Drosophila melanogaster. *PLoS Comput Biol. 2*(5), e51. Epub 2006 May 19.

Pilpel, Y., P. Sudarsanam, and G. M. Church (2001). Identifying regulatory networks by combinatorial analysis of promoter elements. *Nature Genetics 2*, 153–159.

Planta, R., P. Goncalves, and W. Mager (1995). Global regulators of ribosome biosynthesis in yeast. *Biochemistry and cell biology 73*, 825–834.

Quinlan, J. (1996). Bagging, boosting, and c4. 5. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 725–730.

Rajewsky, N., M. Vergassola, U. Gaul, and E. D. Siggia (2002, Oct 24). Computational detection of genomic cis-regulatory modules applied to body patterning in the early drosophila embryo. *BMC Bioinformatics. 3*, 30. Epub 2002 Oct 24.

Ratsch, G., B. Sch
"olkopf, A. Smola, K. M
"uller, T. Onoda, and S. Mika (1999). &nu;-Arc: Ensemble Learning in the Presence of Outliers. In *Advances in Neural Information Processing Systems 12.* Citeseer.

Ratsch, G., B. Scholkopf, A. Smola, K. Muller, T. Onoda, and S. Mika (1999). v-arc: Ensemble learning in the presence of outliers. In *Advances in Neural Information Processing Systems 12.* Citeseer.

Raychaudhuri, S., J. Stuart, and R. Altman (2000). Principal components analysis to summarize microarray experiments: application to sporulation time series. In *Pac Symp Biocomput*, Volume 5, pp. 455–466.

Reinke, V., I. S. Gil, S. Ward, and K. Kazmer (2004). Genome-wide germline-enriched and sex-biased expression profiles in Caenorhabditis elegans. *Development 131*(2), 311–323.

Roe, B., H. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor (2005). Boosted decision trees as an alternative to artificial neural networks for particle identification. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 543*(2-3), 577–584.

Schapire, R. (1990). The strength of weak learnability. *Machine learning 5*(2), 197–227.

Schapire, R. (1999). Theoretical views of boosting and applications. In *Algorithmic Learning Theory: 10th International Conference, ALT'99, Tokyo, Japan, December 1999. Proceedings*, pp. 13. Springer.

Schapire, R., Y. Freund, P. Bartlett, and W. Lee (1998a). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics 26*(5), 1651–1686.

Schapire, R. E., Y. Freund, P. Bartlett, and W. S. Lee (1998b, October). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics 26*(5), 1651–1686.

Segal, E., M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman (2003a). Module networks: Identifying regulatory modules and their condition specific regulators from gene expression data. *Nat Genet 34*(2), 166–176.

Segal, E., M. Shapira, A. Regev, D. Pe'er, D. Botstein, D. Koller, and N. Friedman (2003b). Module networks: identifying regulatory modules and their condition-specific regulators from gene expression data. *Nature genetics 34*(2), 166–176.

Shannon, P., A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker (2003, Nov). Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res 13*(11), 2498–504.

Shim, Y. (1999). elt-1, a gene encoding a caenorhabditis elegans GATA transcription factor, is highly expressed in the germ lines with msp genes as the potential targets. *Mol Cells 9*, 535–541.

Subramanian, A., P. Tamayo, V. Mootha, S. Mukherjee, B. Ebert, M. Gillette, A. Paulovich, S. Pomeroy, T. Golub, E. Lander, et al. (2005). Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proc Natl Acad Sci U S A 102*(43), 15545–15550.

Taddei, A., G. Van Houwe, S. Nagai, I. Erb, E. Van Nimwegen, and S. Gasser (2009). The functional importance of telomere clustering: Global changes in gene expression result from SIR factor dispersion. *Genome research 19*(4), 611.

Teng, Y., L. Girard, H. B. Ferreira, P. W. Sternberg, and S. W. Emmons (2004, Dec 15). Dissection of cis-regulatory elements in the C. elegans Hox gene egl-5 promoter. *Dev Biol. 276*(2), 476–92.

ter Braak, C. and S. de Jong (1998). The objective function of partial least squares regression. *Journal of chemometrics 12*(1), 41–54.

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *J R Stat Soc Series B 58*(1), 267–288.

Tompa, M., N. Li, T. L. Bailey, G. M. Church, B. D. De Moor, E. Eskin, A. V. Favorov, M. C. Frith, Y. Fu, J. J. Kent, V. J. Makeev, A. A. Mironov, W. S. Noble, G. Pavesi, G. Pesole, M. Régnier, N. Simonis, S. Sinha, G. Thijs, J. V. van Helden, M. Vandenbogaert, Z. Weng, C. Workman, C. Ye, and Z. Zhu (2005, January). Assessing computational tools for the discovery of transcription factor binding sites. *Nat. Biotechnol. 23*(1), 137–144.

Valiant, L. (1984). A theory of the learnable. *Communications of the ACM 27*(11), 1134–1142.

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

Warmuth, M., J. Liao, and G. Rätsch (2006). Totally corrective boosting algorithms that maximize the margin. In *Proceedings of the 23rd international conference on Machine learning*, pp. 1001–1008. ACM New York, NY, USA.

Waterston, R., K. Lindblad-Toh, E. Birney, J. Rogers, J. Abril, P. Agarwal, R. Agarwala, R. Ainscough, M. Alexandersson, P. An, et al. (2002). Initial sequencing and comparative analysis of the mouse genome. *Nature 420*(6915), 520–562.

Watson, J. and F. Crick (2003). A structure for deoxyribose nucleic acid. *A century of Nature: twenty-one discoveries that changed science and the world*, 82.

Way, J. C. and M. Chalfie (1988). mec-3, a homeobox-containing gene that specifies differentiation of the touch receptor neurons in C. elegans. *Cell 54*(1), 5–16.

Weinberger, K. Q., F. Sha, Q. Zhu, and L. K. Saul (2007). Graph laplacian regularization for large-scale semidefinite programming. In B. Schölkopf, J. Platt, and T. Hoffman (Eds.), *Advances in Neural Information Processing Systems 19*, pp. 1489–1496. Cambridge, MA: MIT Press.

Whiteson, D. and N. Naumann (2003). Support vector regression as a signal discriminator in high energy physics. *Neurocomputing 55*(1-2), 251–264.

Widom, J. (1998). Structure, dynamics, and function of chromatin in vitro. *Annu Rev Biophys Biomol Struct. 27*, 285–327.

Wold, H. (1966). Estimation of principal components and related models by iterative least squares. *Multivariate analysis 1*, 391–420.

Wold, H. (1975). *Path models with latent variables: the NIPALS approach.* Acad. Pr.

Wold, S., A. Ruhe, H. Wold, and W. Dunn III (1984). The collinearity problem in linear regression. The partial least squares (PLS) approach to generalized inverses. *SIAM Journal on Scientific and Statistical Computing 5*, 735.

Xue, D., M. Finney, G. Ruvkun, and M. Chalfie (1992). Regulation of the mec-3 gene by the C.elegans homeoproteins UNC-86 and MEC-3. *EMBO J 11*(13), 4969–4969.

Yamamoto, A., Y. Mizukami, and H. Sakurai (2005). Identification of a novel class of target genes and a novel type of binding sequence of heat shock transcription factor in Saccharomyces cerevisiae. *Journal of Biological Chemistry 280*(12), 11911.

Yeung, M. K., J. Tegner, and J. J. Collins (2002). Reverse engineering gene networks using singular value decomposition and robust regression. *Proceedings of the National Academy of Science 99*, 6163–8.

Zellner, A. and H. Keuzenkamp (2001). *Simplicity, inference and modelling: keeping it sophisticatedly simple.* Cambridge Univ Pr.

Zhang, N. R., M. C. Wildermuth, and T. P. Speed (2008). Transcription factor binding site prediction with multivariate gene expression data. *Ann. Appl. Stat. 2*(1), 332–365.

Zhang, Y., C. Ma, T. Delohery, B. Nasipak, B. C. Foat, A. Bounoutas, H. J. Bussemaker, S. K. Kim, and M. Chalfie (2002, Jul 18). Identification of genes expressed in C. elegans touch receptor neurons. *Nature. 418*(6895), 331–5.

Appendix A

Proof for D_{KL} on PSSMs

Consider two PSSMs p and q of equal length l defined for nucleotide sequences $x_1, ..., x_l$. The probability of observing $x_1, ..., x_l$ is $p(x_1, ..., x_l) = \prod_{i=1}^{l} p_i(x_i)$ given PSSM p and $q(x_1, ..., x_l) = \prod_{i=1}^{l} q_i(x_i)$ given PSSM q. The Kullback-Leibler divergence between p and q is equal to

$$D_{KL}(p||q) = \sum_{x_1,..,x_l} p(x_1, ..., x_l) log \frac{p(x_1, ..., x_l)}{q(x_1, ..., x_l)}$$

where the summation goes over all possible sequences $x_1, ..., x_l$ with $x_i \in \{A, C, G, T\}$ for all i. Since the positions are independent, we get

$$\begin{aligned} D_{KL}(p||q) &= \sum_{x_1,..,x_l} \prod_k p_k(x_k) log \frac{\prod_n p_n(x_n)}{\prod_m q_m(x_m)} \\ &= \sum_{x_1,..,x_l} \prod_k p_k(x_k) (\sum_n log\ p_n(x_n) - \sum_m log\ q_m(x_m)) \\ &= \sum_n \sum_{x_1,..,x_l} \prod_k p_k(x_k) (log\ p_n(x_n) - log\ q_n(x_n)) \end{aligned} \quad (A.1)$$

Using $\sum_{x\in\{A,C,G,T\}} p_n(x) = 1$, we then have

$$\begin{aligned} D_{KL}(p||q) &= \sum_n \sum_{x_n} (log\ p_n(x_n) - log\ q_n(x_n)) \\ &= \sum_n \sum_{x_n\in\{A,C,G,T\}} p_n(x_n) log\ \frac{p_n(x_n)}{q_n(x_n)} \\ &= \sum_n D_{KL}(p_n||q_n) \end{aligned} \tag{A.2}$$

Appendix B

Quadratic approximation of relative entropy

In TotalBoost, we implement *sequential quadratic programming* (SQP) algorithm for solving the convex optimization problem. We initially set the approximate solution to $\hat{w} = w^0$ and optimize $\hat{w}$ in a sequential way. Given the current solution $\hat{w}$, we determine the update w by minimizing the 2nd order Taylor approximation of change in relative entropy

$$\begin{aligned}
\triangle(w, w^0) - \triangle(\hat{w}, w^0) &= \sum_{ge} w_{ge} \log \frac{w_{ge}}{w^0_{ge}} - \sum_{ge} \hat{w}_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}} \\
&= \sum_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}} (w_{ge} - \hat{w}_{ge}) + w_{ge} (\log \frac{w_{ge}}{w^0_{ge}} - \log \frac{\hat{w}_{ge}}{w^0_{ge}}) \\
&= \sum_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}} (w_{ge} - \hat{w}_{ge}) + w_{ge} \log (1 + \frac{(w_{ge} - \hat{w}_{ge})}{\hat{w}_{ge}}) \\
\text{2nd order Taylor expansion} &= \sum_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}} (w_{ge} - \hat{w}_{ge}) + w_{ge} (\frac{(w_{ge} - \hat{w}_{ge})}{\hat{w}_{ge}} - \frac{1}{2} \frac{(w_{ge} - \hat{w}_{ge})^2}{\hat{w}^2_{ge}})
\end{aligned}$$

$$\text{since} \sum_{ge} w_{ge} = \sum_{ge} \hat{w}_{ge} = 1$$

$$\begin{aligned}
&= \sum_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}}(w_{ge} - \hat{w}_{ge}) + w_{ge}(\frac{(w_{ge} - \hat{w}_{ge})}{\hat{w}_{ge}} - \frac{1}{2}\frac{(w_{ge} - \hat{w}_{ge})^2}{\hat{w}^2_{ge}}) \\
&= \sum_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}}(w_{ge} - \hat{w}_{ge}) + \frac{(w_{ge} - \hat{w}_{ge})^2}{\hat{w}_{ge}} - \frac{1}{2}\frac{w_{ge}(w_{ge} - \hat{w}_{ge})^2}{\hat{w}^2_{ge}} \\
\text{remove 3rd oder term} &= \sum_{ge} \frac{1}{2\hat{w}_{ge}}((w_{ge} - \hat{w}_{ge})^2 + 2\hat{w}_{ge} \log \frac{\hat{w}_{ge}}{w^0_{ge}}(w_{ge} - \hat{w}_{ge}))
\end{aligned}$$

Appendix C

Glossary

AdaBoost - Adaptive Boosting

ADT - Alternating Decision Tree

amino acid - structural unit of a protein

base pair - pair of nucleotides in DNA

boosting - machine learning algorithm typically for classification

Caenorhabditis elegans - specific worm species

CG - column generation method for solving difficult large-scale optimization problems

classification - prediction of a class label based on input data

clustering - grouping examples into clusters based on data patterns

concensus - sequence model for a motif

DNA - deoxyribonucleic acid

edge - one measure of the performance of weak rule with respect to example weight distribution

Gene expression - mRNA level

generalization error - expected error of a prediction function on a data distribution

GO - Gene Ontology

graph-mer - graphs of k-mers that predict gene expression trajectories

IS - iteration score

Latent factor - variables that are extracted to account for predictor variation and model the responses well as well

LPBoost - Linear Programming Boosting

margin - one measure of confidence in prediction

margin-based score - assessment of impact of a motif on the confidence of predictions on a set of training examples

MEDUSA - Motif Element Discrimination Using Sequence Agglomeration

MCS - motif conservation score based on the conservation rate of k-mer in the promoter region

microarray - chip for measuring gene expression (mRNA levels)

motif - sequence pattern for a TF binding site

overfitting - a statistical model describes noise in the data instead of the underlying relationship

PCA - a statistical technique converting many highly correlated variables into a set of uncorrelated variables called principal components

PLS regression - a regression method combining dimension reduction with multivariate regression

promoter sequence - non-coding DNA region for transcription factor binding

PSSM - position-specific scoring matrix

relative entropy - non-symmetric measure of the difference between two probability distributions

Saccharomyces cerevisiae - specific yeast organism (baker's yeast)

supervised - type of learning problem based on examples with labels (e.g. classification)

transcription factor - protein binding to DNA regulating expression of target genes

test loss - empirical error on test data

TRANSFAC - database for transcription factor binding sites

TotalBoost - Totally Corrective Boosting with relative entropy regularization

unsupervised - type of learning problem based on examples without labels (e.g. clustering)

weak rule - prediction function producing an error better than random guessing