Graph-based Regularization in Machine Learning: Discovering Driver Modules in Biological Networks

Xi Gao

Graph-based Regularization in Machine Learning: Discovering Driver Modules in Biological Networks

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

by

Xi Gao

Master of Science in Bioinformatics from Virginia Commonwealth University, 2010
Bachelor of Science in Medicine from Sun Yat Sen University, 2008

Director: Dr. Tomasz Arodz
Assistant Professor, Department of Computer Science

Virginia Commonwealth University
Richmond, Virginia
May, 2015

Contents

List of Tables

List of Figures

List of Algorithms

Abstract

Graph-based Regularization in Machine Learning:

Discovering Driver Modules in Biological Networks

by

Xi Gao

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University

Virginia Commonwealth University, 2015

Director: Dr. Tomasz Arodz, Department of Computer Science

Curiosity of human nature drives us to explore the origins of what makes each of us different. From ancient legends and mythology, Mendel's law, Punnett square to modern genetic research, we carry on this old but eternal question. Thanks to technological revolution, today's scientists try to answer this question using easily measurable gene expression and other profiling data. However, the exploration can easily get lost in the data of growing volume, dimension, noise and complexity. This dissertation is aimed at developing new machine learning methods that take data from different classes as input, augment them with knowledge of feature relationships, and train classification models that serve two goals: 1) class prediction for previously unseen samples; 2) knowledge discovery of the underlying causes of class differences. Application of our methods in genetic studies can help scientist take advantage of existing biological networks, generate diagnosis with higher accuracy, and discover the driver networks behind the differences. We proposed three new graph-based regularization algorithms. Graph Connectivity Constrained AdaBoost algorithm combines a connectivity module, a deletion function, and a model retraining procedure with the AdaBoost classifier. Graph-regularized Linear Programming Support Vector Machine integrates penalty term based on submodular graph cut function into linear classifier's objective function. Proximal Graph LogisticBoost adds lasso and graph-based penalties into logistic risk function of an ensemble classifier. Results of tests of our models on simulated biological datasets show that the proposed methods are able to produce accurate, sparse classifiers, and can help discover true genetic differences between phenotypes.

Chapter 1

Introduction

1.1 Motivation

In the past two decades, techniques for exploration of biological systems grew swiftly. Gene expression profile measurement became easier, faster and cheaper. Array-based methods such as DNA microarrays and sequence-based techniques like RNA-seq are taking the main stage in biological and medical experiments. Other techniques that operate at genetic, epigenetic, transcriptomic, proteomic and metabolomic level are also in rapid progress. In result, scientists can capture global snapshots of thousands gene expression levels of many patients. Increasingly, they are also able to place them in the context of gene mutations, promoter methylation, chromatin states, protein abundances, and lipid profiles.

Rapidly growing volumes of experimental data are full of promise, but snapshots with so many variables pose a challenge – they are no longer easily interpretable. Results from profiling experiments are increasingly easy to obtain, but become notoriously hard to process and understand. The purposes of gene expression and other profiling analyses is to formulate novel and plausible hypotheses to narrow down possible new knowledge of studied objects, such as the explanations of a certain disease. Then, further detailed experiments focusing on narrowed list of targets should be performed to confirm the hypotheses. However, due to the scale, noise and complexity of profiling data, formulating reasonable hypotheses becomes a major bottleneck.

Machine learning field of computer science has a long history of relationships with biological research. Concepts from biology inspired many machine learning algorithms such as artificial neural network, likewise the broad application of machine learning algorithms benefits many biology fields, especially genetic research. By their nature, machine learning algorithms learn from existing data to derive models for prediction, classification and pattern recognition. This facilitates the process of hypotheses generation and reasoning about molecular mechanisms in many areas of genetic research, including high-throughput gene expression analysis. In unsupervised studies, clustering algorithms group genes or patients based on the distances calculated from the expression matrix. In supervised studies, classification models with high prediction accuracy can serve as diagnostic or prognostic rules. Compact models with selected small number of variables, or features, can help scientists narrow down from countless possible hypotheses to a selected few, and raise evidence to illustrate the underlying basis of the molecular differences between phenotypes.

Despite many advances, traditional machine learning algorithms struggle with forming models that are accurate, interpretable and concise when faced with highly-dimensional profiling data. One reason is that classification methods based on statistical learning are purely data-driven. They do not use existing knowledge when they analyze the data. They miss on the rapidly growing domain knowledge related to genetic research – gene ontologies, gene regulatory interactions, protein-protein interactions. Many databases with useful information are available online and a large number of them are organized in the form of biological network. Can machine learning algorithms be designed to comprehend domain knowledge and use it for analyzing gene expression data? Can such new algorithms improve classification performances? Driven by these questions, we steered our research towards designing algorithms for incorporating domain knowledge in the form of biological networks into machine learning methods in a way that leads to classification models with improved accuracy and interpretability.

There is an abundant knowledge of gene regulatory and protein-protein interactions, but what are the potential benefits of using biological networks in model training? First, in supervised machine learning, selecting informative features for training is the key to building successful models. The immense number of features in the analysis of gene profiles makes many learning methods lose power. The nature of graph-based biological networks illustrate the relationships between genes - these relationships reveal closeness and dependency between features, and can counter the effects of statistical noise inherent in datasets with small number of samples. Combined with methods from graph theory, these feature-to-feature relationships could be used in feature grouping, extraction and selection. For example, grouping of genes into sets, based on pathways in a network or other information, resulted in the move from univariate analysis to gene set enrichment analysis. In machine learning, we could use feature relationships as a source of constraints on models, to guide training and optimization. Second, previous domain knowledge could help rationalize experiment design. Gene expression analysis used to focus on single gene expression variation between phenotypes. However, when genetic research provide more gene regulation information in terms of network, many researchers start differential co-expression analysis [28, 29, 60] that leads to fruitful result. Third, models built from biological networks are easier to interpret. In univariate gene differential expression analysis or traditional supervised feature selection studies, multiple single selected genes leave scientists no biological mechanism to follow up on, but network-guided expression analysis select significant changed gene subnetworks or pathways to be explored further. Fourth, limiting the models to subnetworks can lead to simpler models. What are the potential benefits of choosing simpler classifiers? Simple models mean shorter testing time, so given the same accuracy, simpler models are more efficient. Also, simple models are less prone to overfitting. Furthermore, as we stated before, smaller connected models are easier to interpret biologically.

1.2 Contributions of the Dissertation

In this study we propose three method for constructing classifiers that predict the phenotype of samples. Even though many state-of-art methodologies including machine learning intend to construct systems to mine out the differences between phenotypes through gene expression, there is shortage of methods that build high-accuracy classifiers that can facilitate biology elucidation.

Our first method is Graph Connectivity Constrained AdaBoost. By adding network connectivity constraint, deletion function and model retraining procedure to Adaboost, we constructed a new supervised machine learning classification algorithm. We tested our method on simulated data with known differences between classes. Our proposed method shows higher predicting accuracy, flexibility and interpretability than state-of-the-art methods. Our second method is Graph Regularized Linear SVM. By adding graph regularization term in to linear SVM objective function, we encourage linear SVM to select features that organize small highly internally connected independent subnetworks. We test our graph regularized linear SVM with other linear SVM methods on three biology datasets and three non-biology datasets. Results show our Graph Regularized Linear SVM method outperform other linear SVM methods. Our third method is Proximal Graph LogisticBoost. We modify the risk function of LogisticBoost by adding convex extension of submodular graph cut set function and integrating it into boosting through proximal gradient descent. Proximal graph LogisticBoost highly improved the performance of AdaBoost and LogisticBoost in biological datasets.

In summary, the major contributions of the research are

1. We integrated biological network knowledge with machine learning algorithms for gene expression analysis.
2. We proposed three new graph-based supervised classification methods that has been shown to:
   (a) predict sample classes with high accuracy;
   (b) produce sparse classifiers;
   (c) discover the true genetic differences between different phenotypes.
3. We improved performance of AdaBoost with our Graph Connectivity Constrained AdaBoost by including:
   (a) network connectivity penalty module;
   (b) deletion function;
   (c) variable importance feature selection model retraining procedure.
4. We improved performance of linear SVMs with our Graph Regularized Linear SVM by including:

   (a) graph regularization term into SVM objective function;

5. We improved boosting with our Proximal Graph LogisticBoost by including:

   (a) graph regularization term into logistic risk function;

   (b) lasso regularization term into logistic risk function;

6. More generally, we demonstrated how to incorporate nonlinear, non-differentiable penalty terms, e.g. those based on submodular set functions, into boosting.

1.3 Structure of the Dissertation

The rest of this dissertation is organized in the following way. In Chapter 2 we review the background relevant to our research. Specifically, in Section 2.1 we explore major types and sources of biological networks. In Section 2.2 we go through the major challenges of gene expression analysis and summarize the state-of-the-art methods that approach the problem from different angles. In Section 2.3 we recapitulate basic knowledge of machine learning that is the foundation of our research.

In Chapter 3, we introduce the experimental framework and the datasets we will use in evaluating the proposed methods in subsequent chapters. In Section 3.1 we explain the cross-validation setup we used. Also, we introduce the metrics that measure the performance of classification methods. In Section 3.2 and Section 3.3, we describe three non-biological datasets and three biological datasets we used for evaluating our methods.

In Chapter 4 we introduce a new machine learning classification algorithm we developed for solving the problem of graph-based learning. We introduce classical AdaBoost algorithm that our algorithm is based on in Section 4.1. Then, in Section 4.2 we describe the design of the new algorithm, Graph Connectivity Constrained AdaBoost, and the details of the connectivity penalty module, deletion function and the model retraining procedure. Connectivity module integrates biological network knowledge into model training, while deletion function and retraining procedure lead to smaller classifiers of higher accuracy. We analyze our method and show their performance in Sections 4.3 and 4.4.

In Chapter 5, we introduce graph regularization in linear SVM. In Section 5.1 we introduce linear SVM and the convex extensions of submodular functions that will form the basis for the graph regularization. In Section 5.2, we propose our Graph Regularized Linear SVM method. In Section 5.3 we present results of our proposed method along with other linear SVM methods.

In Chapter 6, we introduce our submodular graph regularized boosting method. We explain boosting theory in the perspective of gradient descent in Section 6.1. We elaborate on details of the new Proximal Graph LogisticBoost algorithm in Section 6.2. Then we compare the new graph-based boosting method results along with state-of-art results in Section 6.3.

Finally, in Chapter 7, we summarize the conclusions from our current solutions to the problem of classification and finding differentially expressed modules in biological networks.

Chapter 2

Background

Our work is focused on differential analysis of molecular profiles, such as gene expression data, by using machine learning methods, and by extending them with knowledge of biological networks. In this Chapter, we provide background information about these three focus areas.

2.1 Biological Networks

Graph is a data structure broadly used to represent objects as well as their relationships. Biological objects such as genes and proteins organized in a graph form a biological network. In bioinformatics, this typically includes gene regulatory networks, protein-protein interaction networks, signaling pathway networks or metabolic networks. Depending on the type of the interactions, these networks can be classified as a directed or undirected graphs. As the knowledge of biochemistry and molecular biology accumulates, biological networks become more and more complex. Large, comprehensive networks with large number of nodes and edges provide scientists opportunities, but also challenges. How to efficiently store and organize these networks and mine out the hidden biological information behind the labyrinthine graphs are two major issues. Various databases are built for biological networks and many existing and new graph algorithms have been used in conjunction with these networks to answer biological questions.

Here we review two major biological networks we have been using in our research - BioGRID and PhosphoNetworks. BioGRID is a public database that contains comprehensive information on genetic and protein-protein interactions collected from multiple sources for human and other model organisms [49]. It was first created in 2003 and it is continuously updated [44]. There are 217,928 human genetic and physical interactions in total in the 2013 version of BioGRID network we downloaded. Most BioGRID network interactions are gathered from literature studies, and the experimental method used for discovering the interaction is recorded in BioGRID along the interactions. A study based on Bayesian learning showed differences in confidence that can be assigned to interactions discovered by different experimental system [58]. This difference in confidence of evidence is presented as probability scores, where higher score means higher confidence in the evidence behind the interaction.

Another biological network we used is focused on a subset of protein-protein interactions. PhosphoNetworks is a public biological network database that focuses on human kinase-substrate phosphorylation relationships [22, 23, 38]. Rather than being collected from literature, all the interactions stored in PhosphoNetworks are validated by proteomic-based strategy. Phosphorylation reactions of 289 human kinases on 4,191 human proteins are tested and 3,656 phosphorylation relationships are identified and stored in the database.

2.2 Differential Analysis of Gene Expression

As genome-wide expression profiles became more available, computational methods aimed at discovering differences between two or more phenotypes entered into a rapid development phase to deal with the newly available breadth of data. Because of the relatively small sample size and large feature space of genome-wide expression data, traditional univariate analysis based on comparison of means or correlation with class variable showed multiple disadvantages:

1. low statistical power resulting from ignoring the fact that groups of genes act together in a biological function and mild change of expression level of each gene could result in significant cell malfunction or hyper-function that leads to phenotype differences. Treating genes independently reduces the ability to identify genes that act as a set but with non-significant expression level changes by themselves;
2. no individual statistically significant gene may be identified after multiple hypothesis

correction for large number of variables;

3. if some genes are identified as significant, there is often little overlap of genes in experiments conducted by independent groups;
4. identified genes cannot tell us much about the biological processes and pathways involved in the studied pathology.

Various categories of methods are designed to overcome these disadvantages of traditional univariate methods. Below, we review the major current solutions for the above problems.

2.2.1 Differential Co-expression

Differential co-expression analysis is gaining attention in recent years as a tool for finding phenotypic differences that are more complex than univariate differences. The approach aims at detecting pairs of genes that show differences in gene regulation and cellular signaling between two phenotypes. Differential co-expression brings a new perspective on differences between phenotypes – that a specific phenotype could result from differences in gene regulation that do not significantly alter average expression levels of genes, but alter the pattern of behavior of the genes in tandem. For example, expression levels of a pair of genes may be tightly correlated in one phenotype, where one of the genes regulates the other, but uncorrelated in another phenotype, where the regulation is lost due to mutation or other reasons. This idea has been proven true in a lot of biological conditions such as obesity [52], aging [48] and cancer [57].

Existing differential co-expression detection methods usually measure some gene pair relationship score separately in each phenotype condition, then compare the difference of these scores. The score most often used is the Pearson correlation coefficient, applied in conjunction with Fisher's Z transformation [7, 61] that yields a probability distribution of the differences in correlation. However, Pearson correlation coefficient is highly affected by outlying samples. Efforts to make correlation more resilient to outliers were undertaken by using biweight midcorrelation [25]. Rank-based and entropy-based measurements [31], as well as F-statistic [29] are also used. In terms of the number of gene as a unit in the differential co-expression network, some methods aim at gene pairs while others aim at gene modules [28, 40, 60].

2.2.2 Gene Set Enrichment Analysis

Gene Set Enrichment Analysis (GSEA) [50] is designed to improve the disadvantages of traditional statistical methods. GSEA takes genes in a pre-defined gene set as a group and uses it to assess the results of univariate tests. In effect, GSEA translates results from the level of independent genes to a level of pre-defined, biologically meaningful sets of genes. These pre-defined gene sets could be derived from existing biological knowledge. Genes acting together in the same cellular process or same signaling pathway could be grouped together. For example, known oncology genes could be defined as a group. By doing this, genes functioning as a group could be tested together, revealing the cellular function changes associated with disease phenotypes. Based on the list of genes sorted according to the correlation between their expression levels and sample classes obtained from a statistical test, GSEA defines an enrichment score. GSEA operates by going through the sorted list of genes, increasing a score for each gene from the pre-defined set and decreasing it for genes from outside the set. The increment is proportional to the correlation between the gene's expression level and the sample class, so if genes from the set are concentrated at the top or at the bottom of the list, the GSEA score will be far from null. Significance level of the enrichment score is calculated by permuting genes and adjusted for multiple hypotheses testing using false discovery rate (FDR). GSEA was originally validated on leukemia and lung cancer data sets and the results showed the method is able to discovery strong significant gene sets while univariate analysis find little significant gene and no overlap between repeat experiments [50].

Gene set enrichment analysis, even though it overcomes a lot of disadvantages of traditional univariate analysis, has its limitations. It only tests genes in a set together as a whole but fails to take the topology of inter-gene relationships into consideration. Moreover, as it is only based on pre-defined gene sets, it has no power to discover new gene sets that differentiate biological phenotypes.

2.2.3 Differential Sub-network Detection

Based on observed data, can we tell which genes changed expression as a group under different experimental conditions and what are the signaling pathways connecting these genes? In order to move beyond pre-defined pathways employed in gene set enrichment analyses and be able to answer questions like this, research in computational biology started to focus on methods for detecting active modules under certain experiment conditions. Ideker and his team was the first one who started to tackle this problem [24]. They first build up a statistical scoring system for sub-networks to detect how much the expression level of such sub-network changed under certain condition. Then, using simulated annealing, they search the full network to find the sub-network with the optimal score heuristically. After obtaining high-scoring sub-networks of genes, they used Fisher's exact test and random walk approach to determine statistical significance of discovered sub-networks. Cytoscape, a commonly used software in network biology, employed this sub-network searching algorithm as a plug-in, ActiveModules, and it has been applied in many biological domains, such as diabetes [32]. The ActiveModules sub-network searching algorithm has been extended by combining it with differential co-expression method, thus replacing genes by gene interactions as the input [17]. Rather than picking up the sub-network of genes, the methods intends to pick up the sub-network of gene-interactions that changed most under experiment conditions. Pearson correlation coefficient is used to represent the edge scores of pairs of genes, and again simulated annealing is used to pick up the optimal sub-network of gene edges with the most statistically significant accumulated edge score.

Chuang et al. [8] used a greedy approach for gene sub-network discovery and applied it in breast cancer metastasis research. In their algorithm, they give an activity score to each gene sub-network by averaging gene expressions of such sub-network. Then they calculate mutual information of activity score of each sub-network and sample classes. They take each single gene as a seed, and grow corresponding sub-network using such seed greedily. Sub-network with local maximum mutual information score will finally be chosen.

Most gene sub-network detection methods try to search for connected gene groups that, in aggregate, exhibit statistically significant differential expression. Another approach to this problem is to use an additive score for the group, based on the univariate scores of individual genes. Then, the task is to find a connected component from a graph that maximizes the score. With non-negative scores, the maximum is the whole graph, so a penalty is used to limit the size of the discovered subnetwork. If the penalty is assigned to each edge in the subnetwork, the optimal solution is a spanning tree, known as the prize-collecting Steiner tree. Steiner tree problem is NP-hard and has been studied extensively [11, 27, 36, 59]. Based on Dreyfus-Wagner algorithm [11], Scott et al. [46] developed their Steiner tree algorithm to pick up meaningful gene sub-network. They calculate p-values of differential expression for each gene and used $-\log(1-p)$ as prize for each gene. Based on a known gene network structure and user-defined gene prizes, their algorithm could pick up the sub-network that represents regulatory pathway altered under experimental condition. Bailly-Bechet et al. approached the sub-network detection problem the same way with an improved solver for the Steiner tree problem [2].

2.3 Machine Learning

Many of the most successful methods for analyzing data coming from biological experiments involve machine learning. Machine learning is an important discipline in computer science. It emerged from early artificial intelligence field and touches many different fields of science and industry, such as biology, neuroscience, image processing, robotics and language processing. The main task of machine learning is to learn, recognize and predict patterns from data. By applying machine learning algorithms, we turn raw records into new knowledge. Arhur Samuel defined machine learning in 1959 as a "Field of study that gives computers the ability to learn without being explicitly programmed". This points to the generalization ability of machine learning algorithms. A good framework of supervised machine learning algorithm should be autonomous. Given training data and domain knowledge, algorithms should be able to derive model for future prediction by integrating information without human interference.

2.3.1 Basic Terms

In our study we focus on supervised learning algorithms. In supervised machine learning, **features** are individual measurable properties or attributes that have been recorded; **samples** are individual objects whose features have been recorded; **classes** are the labels of samples that indicate their categories. Well-behaved features, samples and sample classes are crucial to the success of machine learning algorithms. Good features are supposed to be discriminative and independent. That means there is little overlapped information or association between features. In practice, this ideal is rarely reached. This opens the opportunity for our proposed graph-based regularization methods, which make use of known relationships between features captured in a form of a graph. Some examples of features are age, gender, p53 expression value. Good samples are supposed to be independent and identically distributed. That means first there is little association between samples; second all samples of the same class should be polled from the same population. The number of classes should be small compared to the number of samples. Often, machine learning algorithms are designed for problems where only two classes are present. Such binary sample classes are usually represented by 1/0 or 1/-1. For example class label 1 represent healthy people and class label -1 represent cancer patients. In our study all classes are labeled in 1/-1.

2.3.2 Algorithm Types

Depending on the types of their input and output data, machine learning algorithms could be organized into four major fields - supervised learning, unsupervised learning, semi-supervised learning and reinforcement learning. In **supervised learning**, the training data always come in pairs with data labels - we know the correct answers to sample classes. Supervised learning algorithms take samples values and sample classes as input, learn the associations between them in training stage, then try to predict sample classes for new sample values in testing stage based on the learned associations. The most popular supervised learning algorithms today are Support Vector Machine (SVM), Artificial Neural Network (ANN), decision tree and ensemble learning (bagging, boosting and random forest).

In **unsupervised learning**, there is no label for any sample, so there is no training and testing stages as in supervised learning. Unsupervised learning algorithms aim at recognizing the patterns and inner structure of data. For example, ***clustering algorithms*** group samples based on their closeness to each other, while ***association rules*** extract abstract rules of objects' relationships from inventory of object sets. **Semi-supervised learning** is situated between supervised and unsupervised learning. Usually semi-supervised learning data are comprised of large amount of unlabeled and small amount of labeled data.

Reinforcement learning is inspired by behaviorist psychology. Most reinforcement learning algorithms try to learn from previous actions that lead to punishments or rewards so that algorithms' future actions maximize cumulative rewards. It is widely studied in disciplines like game theory, control theory, and operations research. Reinforcement learning models are composed of 1) a set of environment states S, 2) a set of actions A, 3) rules of transitioning between states, 4) rules that determine the scalar immediate rewards of a transition, 5) rules that describe what the agent observes. An example of reinforcement learning application is an autopilot system. We set environment states as different road systems, a set of actions as move forward, backward, or turn, rules to transitioning between different road systems, rules that determine punishments and rewards for wrong or correct operations and rules that describe observed results.

2.3.3 Supervised Learning Algorithms

Our research is focused on supervised learning algorithms. Supervised methods can be categorized based on the properties of data labels. If the data labels an algorithm takes and produces are both continuous, then the algorithm is a **regression algorithm**; otherwise, if the data labels are discrete categorical numbers (usually binarized as 1/0 or 1/-1), then the algorithm is a **classification algorithm**. We focus here on classification algorithms, so below we provide a brief review of the most popular supervised learning algorithms commonly used in classification analysis.

Support Vector Machine (SVM) in its modern form was published in 1995 by Vladimir N. Vapnik and Corinna Cortes [9]. In training phase, SVM looks for a hyperplane that maximizes the margin between two classes of samples. Those samples lying on the margin are called support vectors. Kernel trick in SVM maps samples onto a high-dimensional space that leads to efficient non-linear classification. Soft margin SVM allows for accepting samples lying on the wrong sides of the margin, rendering higher generalization power and limiting overtraining. SVM can be reformulated for regression analysis too. More details about the basic, linear version of SVM will be presented in Chapter 5.1.1.

Artificial Neural Network (ANN) is comprised of input, output and optional hidden layers of nodes that are interconnected between layers. To train an ANN we feed the data through the input layer through hidden layers and collect result from the output layer. By comparing results with sample labels, we calculate errors and correct weights on connections between nodes. Eventually, when all weights converge, a trained neural network system is able to predict sample classes with high accuracy. But ANN is a black box model and its weights are hard to interpret. The first model of ANN was introduced in 1943 [35]. It was once a very popular research topic until 1969, when Marvin Minsky and Seymour Papert's publication put the research into stagnation [37]. They revealed two problems about ANN - single layer neural networks cannot solve "exclusive-or problem"; and multi-layered networks with linear nodes can be reduced to single layer networks. The backpropagation algorithm [43,54] and an ANN model with nonlinear nodes in hidden layers successfully solved the "exclusive-or problem" and neural network regained attention. Even though in the 1990s SVM and other new machine learning methods such as ensemble learning reduced popularity of neural networks, recent research in deep learning [20] resulted in another explosion of interest in the field.

Decision Tree uses a tree-like structure as the classification model. All the training samples pass through the root node, and are directed towards the leaves by decisions at each node. Each node of the tree contains a split based on a threshold of a selected feature. The split guides the samples that reached the node into the node's left and right child. Each of the leaves of the tree indicates a single class to be assigned to samples that reach the leaf. In the training stage we train a decision tree, typically in a greedy fashion, and in the testing stage decision tree could predict sample classes by letting samples go from the root through splits to leaves. Decision tree is not a black box model so it is simpler to understand and interpret. However, as it is not resistant to overfitting and is highly susceptible to noise in the training set, usually decision tree cannot get results with high accuracy in complicated data sets. On the other hand, due to its low computational complexity and inherent instability, decision trees are commonly used as base classifiers in ensemble learning (bagging, random forest, boosting).

Decision Stump is a very simple type of decision tree, in which the tree has just one level. That is, we predict one class for all samples that have the value of the feature higher than the threshold, and the other class for all samples for which the value of the feature is below the threshold. Which feature is the deciding feature, what is the threshold, and which class is above the threshold and which is below is decided based on the training data. We chose one feature, one split or threshold on that feature, and the orientation of that split, in a way that minimizes the classification error on the training set.

Ensemble classifiers aggregate multiple learning models to predict results. This highly enhances the prediction accuracy and model resistance to overfitting. Major ensemble methods are bagging, random forest and boosting. Bagging is also called bootstrap aggregating. It was proposed by Leo Breiman in 1996 [4]. The idea of bagging is to create multiple training sets of the same size by random sampling with replacement from original training set. Models are built upon those new training sets and final prediction is the average of predictions of all the models. Random forest [6, 10] is the application of bagging technique in feature spaces. In addition to sampling samples as in bagging, random forest sample features for multiple model training. Boosting is another ensemble classifier that is superficially similar to bagging, with the added weights of base models, and weights of points that focus training on samples that are hard to classify, typically because they are close to the decision boundary separating the classes. However, it has a deeper interpretation as an iterative method for optimizing the risk of the classifier. Our work is based on extending boosting to improve its accuracy on biological datasets, and we discuss boosting separately in more detail in Sections 4.1.1, 6.1.1 and 6.1.2.

2.3.4 Regularization in Supervised Learning

Classification algorithms such as those described above extract models by learning from training samples data and their categories. Then, the trained model, or a classifier, is used to predict unknown categories for new data samples. Classifiers should generalize well to previously unseen samples, and should also be parsimonious and interpretable.

Many classification problems involve a feature space that is very highly dimensional. For example, in biological classification problems, the number of experimental genes or proteins are almost always much larger than the number of patients. As a result, the density of training samples in the feature space is very low, which may result in overfitting. Overfitting happens when classifiers describe training data too well, and they lose their robustness and generalization ability. Thus, overfitted classifiers usually perform poorly in prediction and are unstable to noise fluctuation. When feature dimensions are much higher than training sample sizes, classifiers are usually overly complex. Complex classifiers that contain excessive number of parameters are often overfitted.

Various methods were invented to deal with overfitting and regularization is one of the most important class of techniques. Regularization aims to limit overfitting by penalizing model complexity. L_1 and L_2 regularization are classic regularization paradigms that were initialy used in regression, but are also used in classification.

L_2 regularization adds L_2 norm of parameter weights into loss function to control model complexity. For example, in a linear model, the sum of squares of weights of features in the model would be incorporated into the loss function. Ridge regression [21] is a statistical application of L_2 regularization. Classical linear SVM is another example. However, L_2 regularization's feature weight decay effect cannot effectively reduce feature numbers, since penalty drops very quickly as feature weights approach zero, and there's little incentive to actually reach null feature weight and eliminate the feature from the model completely. As a result, classical linear SVM classifiers can still contain many parameters with small weights.

To obtain more parsimonious classifiers, feature selection methods such as recursive feature elimination [19] are usually combined with SVM. These external feature selection methods are called wrapper. They rank features by their importance and remove less important features to reduce feature size. By testing numerous feature sizes, wrappers choose the best feature subset for model training. Wrappers are very computational expensive and unstable due to its greedy disposition [18]. One would desire a regularization method with embedded feature selection property that does feature selection and model construction simultaneously. L_1 regularization can accomplish such effect by driving some feature weights to zero. This leads to sparse models that are preferred [39]. LASSO [51] and LARS [14] are example of statistical methods involving L_1 regularization. Zou and Hastie proposed Elastic net regularization [64] that combines L_1 and L_2 penalty parts. The L_1 penalty part contributes to the sparsity of models and L_2 penalty partially releases $L_1$from its feature selection limitations. Additionally, L_2 penalty part promotes a grouping effect that encourages selection of related features as a group.

The methods described above are all purely data-driven. Besides experimental data, domain knowledge is helpful to model success. For example, in genetic study, gene groupings or interactions established through decades of biology experiments can play an important role in illuminating new studies. In image processing, neighboring pixels are often similar, with variation that can be attributed more to noise than to the actual difference. In recent years, group LASSO was introduced to incorporate additional information about groups formed by features. Also, pre-defined quadratic spatial kernels over features were used to improve image processing using AdaBoost [56]. Quadratic penalties on graphs were also proposed for linear regression [30].

Chapter 3

Evaluation of Supervised Machine Learning Methods

In this Chapter we describe how the methods we propose in this dissertation will be evaluated. This includes the description of performance metrics and the validation procedures. We also present the datasets that will be used for evaluating the methods, and propose our own method for creating realistic, simulated biological two-class datasets.

3.1 Evaluation Scheme

To evaluate supervised learning algorithms, we usually partition samples into **training** and **testing** parts. We feed algorithms training samples and their classes as input, and let algorithms learn a model that recognizes the patterns in the data. Then we can generate predicted classes for testing samples using the model learned from training samples and classes. By comparing the predicted test sample classes to the known true test sample classes, we evaluate the algorithm's performance on test data. The most common problem in supervised machine learning algorithms is **overfitting**. It is also called **overtraining**. In supervised machine learning it means the learning algorithms selects too complicated models to fit training data accurately so that the models fit to noise and loose generalization power to predict future data accurately. The opposite side of overfitting is **underfitting**. It is also called **undertraining**. Underfitting means machine learning algorithms select too simple models to fit training data.

In the situation of underfitting, models have equally low predicting power over both training and testing samples.

Cross-validation is mainly used in supervised learning and it is a useful technique to avoid overfitting. It partitions experiment samples into several different parts called "folds" (e.g. five-fold cross validation partitions samples into five parts). In classification problems, each part is supposed to have the same proportion of samples from each class. Cross validation chooses one part of the partitions as "testing set" while the rest serve as "training set". Machine learning algorithms extract models from training sets in "training stage" and predict on testing sets in "testing stage". We could evaluate algorithms by checking their average performance on all different cross-validation training/testing sets derived from given dataset.

3.1.1 Cross-validation Framework

For testing the methods, we store and use the same cross-validation labels for a given dataset. When we use cross validation to test the method performances on a dataset, we have to avoid using different cross-validation partitions of data for different methods, because different data partition introduce noise to the estimate of the method's performance. We generate and save 10 independent five-fold cross validation labels for each dataset as specified in Algorithm 1. We then test methods for any given dataset using cross-validation labels generated as specified in Algorithm 2. We go through five-fold cross validation of this data set when testing performances of methods. For each method, we average the results of those 50 runs.

Algorithm 1 Cross-validation Label Generation

```
for each dataset $d$ do
  for $j \leftarrow run\{1, ..., 10\}$ do
    for $i \leftarrow class\{1, -1\}$ do
      $S_i = (sample \in class)$;
      $S'_i = Permute(S)$;
      for $k \leftarrow \{1, ..., m\}$ do
        $label(S'_i(k)) = 1 + (k \mod 5)$;
      end for
      $label(S'_{i,sort})$ = sort $label(S'_i)$ according to $Index(S_i)$;
    end for
    save five-fold cross validation $label(S_sort)$ for $j_{th}$ run of data $d$ to file as $S_{d,j}$.
  end for
end for
```

Algorithm 2 Cross-validation Framework

```
for each dataset $d$ do
  for $theMethod \leftarrow \{1, ..., number\ of\ methods\}$ do
    for $j \leftarrow run\{1, ..., 10\}$ do
      $label(S_{d,j}) \leftarrow$ sample cross validation label of $j_{th}$ run, dataset $d$;
      for $i \leftarrow fold\{1, ..., 5\}$ do
        $S_{train} \leftarrow label(S_{d,j}) \neq i$;
        $S_{test} \leftarrow label(S_{d,j}) = i$;
        $result_{j,i}(theMethod, d) = \textbf{Method}(S_{train},\ S_{test})$;
      end for
    end for
    $result(theMethod, d) = \sum_{j=1}^{10} \sum_{i=1}^{5} result_{j,i}(theMethod, d)$;
  end for
  $result'(d) =$ sort $result(d)$ according to AUROC value in $result(d)$;
  save $result'(d)$ for all Methods for data d;
end for
```

Given a dataset, we compare the average results of all methods and sort them by their mean AUROC value, which we introduce below in Section 3.1.2. Experiment results are presented in the result and discussion section by the end of each method chapter.

3.1.2 Performance Evaluation

There are many ways to measure the performance of an algorithm. In supervised learning, we assume we know the true label of some test samples so that we could evaluate our algorithm by comparing the predicted values of labels for testing samples to the true ones. A table called confusion matrix or contingency table (Table 3.1) shows different possible scenarios.

True positives (TP) and true negatives (TN) are correctly classified testing samples. False negative (FN) is rejection error, also called type I error. False negatives are true positive samples that are predicted negative. False positive (FP) is acceptance error, also called type II error. False positives are true negative samples that are predicted positive. From the

Table 3.1: Confusion Matrix

Truth \ Prediction	Positive	Negative	Total
Positive	True Positive (TP)	False Negative (FN)	P
Negative	False Positive (FP)	True Negative (TN)	N

contents of the confusion matrix, we can calculate a set of values that are often used to measure the quality of predictions generated by supervised classification machine learning algorithms.

Sensitivity is also called recall. It means how often the algorithm finds out true positive samples. It is an estimate probability that a true positive sample will be predicted correctly.

$$Sensitivity = TP/P = TP/(TP + FN) \tag{3.1}$$

Specificity means how often the algorithm finds true negative samples. It is an estimated probability that a true negative sample will be predicted correctly.

$$Specificity = TN/N = TN/(FP + TN) \tag{3.2}$$

Accuracy means how often the algorithm finds true sample labels. It is an estimated probability that a sample will be predicted correctly.

$$Accuracy = (TP + TN)/(P + N) \tag{3.3}$$

Precision means how often the predicted positive samples are truly positive. It is an estimated confidence of a predicted positive sample.

$$Precision = TP/(TP + FP) \tag{3.4}$$

False positive rate (FP rate) means how often the true negative samples are falsely predicted positive.

$$FPrate = FP/N = FP/(FP + TN) = 1 - Specificity \tag{3.5}$$

False Discovery rate (FDR) means how often the predicted positive samples are falsely predicted.

$$FDR = FP/(FP + TP) = 1 - Precision \tag{3.6}$$

Receiver Operating Characteristics Curve (ROC curve) is used to analyze performance of binary classifiers that return a continuous decision that is later converted into a binary prediction using a threshold. ROC curve is a smooth line connecting dots plotted in the space of FP rate and sensitivity. Each dot represents the value of FP rate and sensitivity of tested algorithm at a given threshold for classification. Area under the ROC curve (**AUC** or **AUROC**) equals to the probability that the classifier will differentiate a randomly chosen true positive from a random chosen true negative sample. Its power is equivalent to the Wilcoxon/Mann-Whitney U rank test.

3.2 Benchmark Non-biological Datasets

While our work is focused on biological network and data, we have identified three datasets from outside of the biological field, as a way to show the generality of our methods.

3.2.1 Gaussian Dataset

We have created an artificial, toy dataset with the size small enough to all be convenient during algorithm development. The dataset has 10000 samples and 100 features, connected by a random graph depicted in Fig 3.1. In the graph, three small connected subgraphs of different topology are identified, labeled as red, green and cyan. We created three datasets corresponding to these three colors. In each dataset features corresponding to one of the three subgraphs are the discriminative features. All other features are Gaussian random noise.

We tested Heuristic Graph AdaBoost and Graph Regularized Linear SVM on three different Gaussian datasets. We used five fold nested cross validation for parameter tuning and testing results.

3.2.2 Time Series Dataset

We have used a real-world time series datasets with 121 samples and 637 features. The dataset describes a two-class problem of differentiating two types of atmospheric lightnings, based on features that describe electromagnetic power density of the atmosphere measured from a satellite in regular time intervals in the sub-microsecond range. The series of such measurements together forms a time series that describes a single lightning event.

In classical classifier training using e.g. SVM or boosting, the information that the features correspond to consecutive time points would be lost. To show how graph-based regularization can help in time series data, we have constructed a graph in which each feature is connected to the feature representing the previous time step. We have used the graph as an additional knowledge in graph-based regularization.

We tested Heuristic Graph AdaBoost and Graph Regularized Linear SVM on time series datasets. We used five fold nested cross validation for parameter tuning and testing results.

3.2.3 Image Analysis Dataset

Our next dataset is incorporated to demonstrate how graph-based regularization can help with image analysis. The NIST (National Institute of Standards and Technology) handwritten digits database that contains Special Database 1 which is sampled from American Census Bureau employees and Special Database 3 which is sampled from American high school students. NIST is a hard dataset for image processing as its training (Special Database 1) and testing (Special Database 3) samples are from different population. Each image of a digit contains 28 by 28 pixels, that is, 784 features. MNIST (Mixed National Institute of Standards and Technology) handwritten digits dataset mixed Special Database 1 and 3 from NIST dataset. It then split the mixed samples into training set of 60,000 images and testing set of 10,000 images. MNIST is, compare to NIST, an easier dataset for image processing, and is widely used for machine learning method testing, including our tests in this dissertation. We

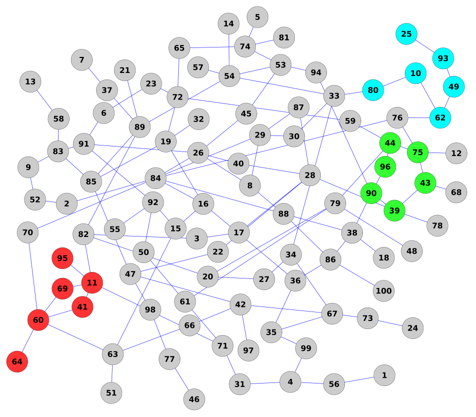

Figure 3.1: Gaussian dataset and its feature network - a random graph with 100 nodes.

have used the pixels in the images as features, and we constructed a regularizing graph by treating each pixel as a vertex, and adding an edge between adjacent pixels. That is, a pixel is connected to it's top, bottom, left and right neighbor.

We tested Heuristic Graph AdaBoost and Graph Regularized Linear SVM on three different Gaussian datasets. We used five fold cross validation for parameter tuning on MNIST training dataset. Then we tested methods on MNIST test set using parameter chosen from previous step.

3.3 Simulated Biological Datasets

We tested Heuristic Graph AdaBoost, Graph Regularized Linear SVM and Proximal AdaBoost on biological datasets. For testing Graph Regularized Linear SVM, we used five fold nested cross validation for parameter tuning and testing results. For Heuristic Graph AdaBoost, we used 10 replicates of two fold cross validation for parameter tuning and testing results. For Proximal AdaBoost, we used five fold cross validation to choose optimal parameters then test on ONE test set.

3.3.1 Motivation for Simulating Biological Datasets

Our research is dedicated to developing new algorithms that detect the underlying differences between phenotypes using gene expression data and gene network information. Gene expression profiles with different phenotypes are organized in a matrix of feature values for each sample and a vector of class labels. Each column of the matrix is a gene acting as a feature while each row is a sample, typically corresponding to a patient. Every sample has a corresponding class label 1 or -1 to indicate which phenotype group it belongs to. Gene network is organized in adjacency matrix that indicates the relationships between features.

The two major tasks of the new algorithms are to 1) obtain models from training data and matching networks to predict classes for previously unseen samples, given sample gene expression profile; 2) extract the hypotheses of underlying differences between two phenotypes from the model. Performance of algorithms with respect to the first task can be assessed using methods such as cross-validation. However, validating the performance of algorithms on the second task is problematic: 1) the true underneath network structures of gene regulation and protein interactions are unknown; 2) the true differences between two phenotypes are unknown. Even though gene expression profiles and network information for real life datasets are widely available, there is only limited information about the causes of differences between phenotypes. Moreover, most available human biological networks are incomplete maps of the genetic interactions. Results of algorithms integrating these incomplete networks with complete genetic expression information are not completely valid or comparable. For the purpose of algorithm development and evaluation, data simulation becomes our best option.

3.3.2 Dataset Simulation Framework

Proper experiment datasets should have gene expression values of two classes from the same network structure. Sample values of class one are generated from the intact network, while sample values of class two are from the network with selected regulation or signaling damage. The regulation damage is achieved by altering the dissociation constant k of corresponding edges. Dissociation constant k of edge $A \to B$ specifies strength of regulation of B by A.

To shorten testing time, we started with networks of limited size but with preserved basic regulation pathways. We extracted desired networks from existing high confidence biological network PhosphoNetworks [23] for simulation. The network extraction details are explained in Subsection 3.3.3, Algorithm 4. All selected impaired edges are connected. We modify dissociation constant k based on network size and type of impairment. Molecular and experimental noises are added to all simulated datasets. Normalization procedure is applied to all noised datasets. We use GeneNetWeaver software [45] to simulate two sample classes of gene expression profiles. The GeneNetWeaver simulation model are illustrated in Subsection 3.3.3.1 and the datasets we obtained from the simulations are described in subsection 3.3.4. Machine learning method performance evaluation criteria are listed in Section 3.1.2. In our research, we use area under the ROC (receiver operating characteristic) curve as the final metric to evaluate all method we test. The complete pipeline for data simulation and method evaluation is outlined in Algorithm 3.

Algorithm 3 Data Simulation Pipeline

```
choose a base network $N$;
$data_{norm} \leftarrow$ GeneNetWeaver simulation from $N$;
define impaired network $N_{impaired}$ with chosen impaired edges $E_{impaired}$ from $N$;
dissociation constant $k'(E_{impaired}) = x \cdot k(E_{impaired})$;
$data_{impaired} \leftarrow$ GeneNetWeaver simulation from $N_{impaired}$ with $k$;
$d = union(data_{norm}, data_{impaired})$
$result(d) = \textbf{crossValidation}(data)$
```

Algorithm 4 Subnetwork Extraction for Simulation

```
Require: : 1) A biological network $G$ with $m$ nodes.
  2) Node size $n$ of desired subnetwork $G'$.
  Select $p$ the longest of all shortest paths in $G$;
  for $i \leftarrow genes\{1, ..., m\}$ do
    $d_i$ = the distance of node $i$ to $p$;
  end for
  $V' \leftarrow$ sort all the vertices $V = (v_1, v_2, ..., v_m)$ in $G$ by corresponding $D = (d_1, d_2, ..., d_m)$;
  select the top $n$ nodes from $V'$;
  $G' \leftarrow$ edges $E'$ in $G$ with both nodes in $V'$.
```

3.3.3 Base Network Extraction

The first step of simulation is to extract a well-structured network of desired size. The extracted networks should preserve some biological pathways and other network properties. Network structures generated from random graph models such as Watts-Strogatz model (small-wold), Barabási–Albert model (scale-free) and Erdős–Rényi model (independent edges) do not capture statistically overrepresented biological network properties such as modularity and network motifs occurrences [42, 47]. Applying raw existing subnetwork extraction methods will break pathways and render graph-based machine learning methods irrelevant. For example, sub-networks of PhosphoNetworks generated from random extraction by GeneNetWeaver have longest path of only 3 edges. Networks for our experiment preferably should have at least one complete long signaling pathway with other pathways growing along it. To achieve this task, we designed an extraction method shown below in Algorithm 4.

3.3.3.1 GeneNetWeaver Simulation Theory

GeneNetWeaver (GNW) [45] is an open-source software for in silico simulation of gene expression data based on an underlying gene regulatory network. Originally it was designed to evaluate network inference methods. Theory behind the simulation model is described in [33] and we summarize it here because it is needed to explain our way of creating different phenotypes.

Expression value simulations are based on a network structure. Given a network, for each gene i, GNW simulates F_i^{RNA}, the change rate of mRNA concentration, and F_i^{Prot}, the change rate of protein concentration. Function $f_i(\cdot)$ serves as the activation function ranging from 0 to 1 representing the relative activation of gene i. Furthermore, m_i is the maximum transcription rate, r_i is the translation rate, λ_i^{RNA} and λ_i^{Prot} are the mRNA and protein degradation rates and $\mathbf{x}, \mathbf{y}$ are vectors containing all mRNA and protein concentration levels, respectively. The dynamical model is

$$F_i^{RNA}(\mathbf{x}, \mathbf{y}) = \frac{dx_i}{dt} = m_i \cdot f_i(\mathbf{y}) - \lambda_i^{RNA} \cdot x_i \tag{3.7}$$

$$F_i^{Prot}(\mathbf{x}, \mathbf{y}) = \frac{dy_i}{dt} = r_i \cdot x_i - \lambda_i^{Prot} \cdot y_i \tag{3.8}$$

Random fluctuations and molecular noise in transcription and translation process are modeled be changing a dynamic model of the form

$$\frac{dX_t}{dt} = V(X_t) - D(X_t) \tag{3.9}$$

into a model

$$\frac{dX_t}{dt} = V(X_t) - D(X_t) + c\left(\sqrt{V(X_t)}\eta_v + \sqrt{D(X_t)}\eta_d\right), \tag{3.10}$$

where $V(X_t)$ is the RNA or protein production; $D(X_t)$ is the degradation; η_v and η_d are independent Gaussian noise processes; and c is a constant to control the amplitude of the noise.

The relationships between a transcription-factor (TF) j and gene i ($j \rightarrow i$) is modeled by the probability of states of gene i. State S_1 means TF j is bound to the promoter region of gene i while state S_0 means not bound. The probability of TF j and gene i binding $P\{S_1\}$ depends on y_j, the concentration of TF j, on k_{ij}, the dissociation constant of the TF from promoter region, and on n_{ij}, the Hill coefficient. The probability can be expressed as

$$P\{S_1\} = \frac{\chi_j}{1+\chi_j} \quad with \quad \chi_j = \left(\frac{y_j}{k_{ij}}\right)^{n_{ij}}. \tag{3.11}$$

If α_0 is the relative activation for state S_0 and α_1 the relative activation for state S_1, given $P\{S_1\}$, we could derive $P\{S_0\}$ as its complement, and define the function describing mean activation of gene i given the concentration y_j of TF j:

$$f(y_j)_i = \alpha_0 P\{S_0\} + \alpha_1 P\{S_1\} = \frac{\alpha_0 + \alpha_1 \chi_j}{1+\chi_j} \tag{3.12}$$

In real biological network, the interaction relationships are more complicated than one to one: one gene could be controlled by N TFs. With each TF possibly bound or not bound to such gene, we will have 2^N states in total. Thus the function describing mean activation of gene i given concentrations y of its N regulators could be calculated as

$$f(\mathbf{y}) = \sum_{m=0}^{2^N-1} \alpha_m P\{S_m\} \tag{3.13}$$

Here is an example how to calculate the activation a gene that is regulated by two TFs:

$$f(y_1, y_2) = \frac{\alpha_0 + \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 \rho v_1 v_2}{1 + v_1 + v_2 + \rho v_1 v_2} \quad with \quad v_j = (y_j/k_j)^{n_j} \tag{3.14}$$

3.3.3.2 Modeling Different Phenotypes

For testing our methods, we need datasets that contain samples from two different phenotypes. To generate such datasets, we need to simulate sample expression values for two phenotypes separately. We first use GNW to simulate normal samples with selected network structure. We choose stochastic simulation with multifactorial molecular and experimental noise added. GNW software will set all simulation parameters automatically. Along with the normal sample expression values, we get a normal network model with parameters from GNW. We create a pathological network model by modifying dissociation constants k of the selected impaired edges and keeping other parameters intact. We feed back this pathological network model to GNW software and it will generate the pathological sample expression values. By doing this, we could ensure samples from both classes come from the same network structure with the same other simulation parameters and same amount of noise added. The only differences are the dissociation constants of impaired edges. We treat these edges and their incident vertices as the “true differences” between two phenotypes. As we can tell from equation 3.11, the modification of dissociation constant k_{ij} of the simulation model changes the regulation strength of TF j over gene i. By altering dissociation constants of selected edges, multiple related gene regulation relationships are impaired in simulated pathological situation.

We applied such modeling procedure to two different network structures - PhosphoFull and Phospho200 network. Because PhosphoFull network has 4375 edges while Phospho200 network has only 591 edges, we increase dissociation constants of the same set of impaired edges 30 times in PhosphoFull network while 2 times in Phospho200 network. To simulate gene mutations of the same pathway, we used a similar modeling procedure on Phospho200 network, but instead of impairing all edges in our predefined set, we create five different mutation models for five different impaired edges from the same set. Each mutation model contains the same set of parameters as the normal network model but with one edge dissociation constant increased 10 times. We simulate one fifth of pathological samples using each mutation model and put these samples together as a whole pathological expression set. This mutation dataset is called Phospho200 Mutation. It models the real-world situation where pathology is based on alteration of a single pathway, but in individual patients different components of that pathway are altered. All datasets are described in details in subsection 3.3.4.

3.3.4 Simulated Datasets used in Experiments

3.3.4.1 PhosphoFull Network Dataset

PhosphoFull network dataset is simulated from base network PhosphoNetworks [23]. PhosphoNetworks is a recently published kinase-substrate network that contains downloadable 1291 human kinases and 4375 kinase-substrate phosphorylation relationships. All the protein prosphorylation interactions recorded in PhosphoNetworks are validated by proteomic methods so all the edges in PhosphoNetworks have uniformed confidence. For class 1, the normal phenotype, we simulate a data matrix of 100 samples for these 1291 gene features.

For class -1, the pathological phenotype, we select the edges between nodes PLK1, CHEK2, EIF2AK2, FYN, MAPK8, increase the dissociation constant k for these edges 30 times in the same simulation model we used for class 1 data. By doing this, we obtain another data matrix of the same dimension for class -1. PhosphoFull network is displayed in Figure 3.2 and the core of the network composed of nodes with both non-zero in and out degrees are magnified in Fig 3.3. Edges with augmented dissociation constants are marked red in both figures and listed in Table 3.2.

Table 3.2: PhosphoNetwork Edges with Modified Dissociation Constant

$PLK1 \rightarrow CHEK2$
$PLK1 \rightarrow MAPK8$
$CHEK2 \rightarrow EIF2AK2$
$EIF2AK2 \rightarrow PLK1$
$EIF2AK2 \rightarrow CHEK2$
$EIF2AK2 \rightarrow EIF2S1$
$FYN \rightarrow EIF2AK2$

3.3.4.2 Phospho200 Network Dataset

As we explained in previous sections, to save experiment time and to have a clearer picture of basic pathway structure of experiment network, we extracted Phospho200 network with 200 nodes from PhosphoNetworks using Algorithm 4. The longest shortest path from PhosphoNetworks is $GSK3A \rightarrow CDK9 \rightarrow CHEK2 \rightarrow EIF2AK2 \rightarrow PLK1 \rightarrow MAPK8 \rightarrow MAPK1 \rightarrow FGR \rightarrow CSNK2A1 \rightarrow EPHA3 \rightarrow MAP3K8 \rightarrow MAP3K14 \rightarrow MAP4K5 \rightarrow CAT$. We extracted 186 nodes which are closest to the longest shortest path, together with the 14 nodes on the longest shortest path form Phospho200 network structure. Figure 3.4 and Figure 3.5 display Phospho200 network structure and its core. Phospho200 dataset simulation procedures are the same as simulation in PhosphoFull dataset in Subsection 3.3.4.1. Dissociation constants are augmented 2 times for impaired edges listed in Table 3.2.

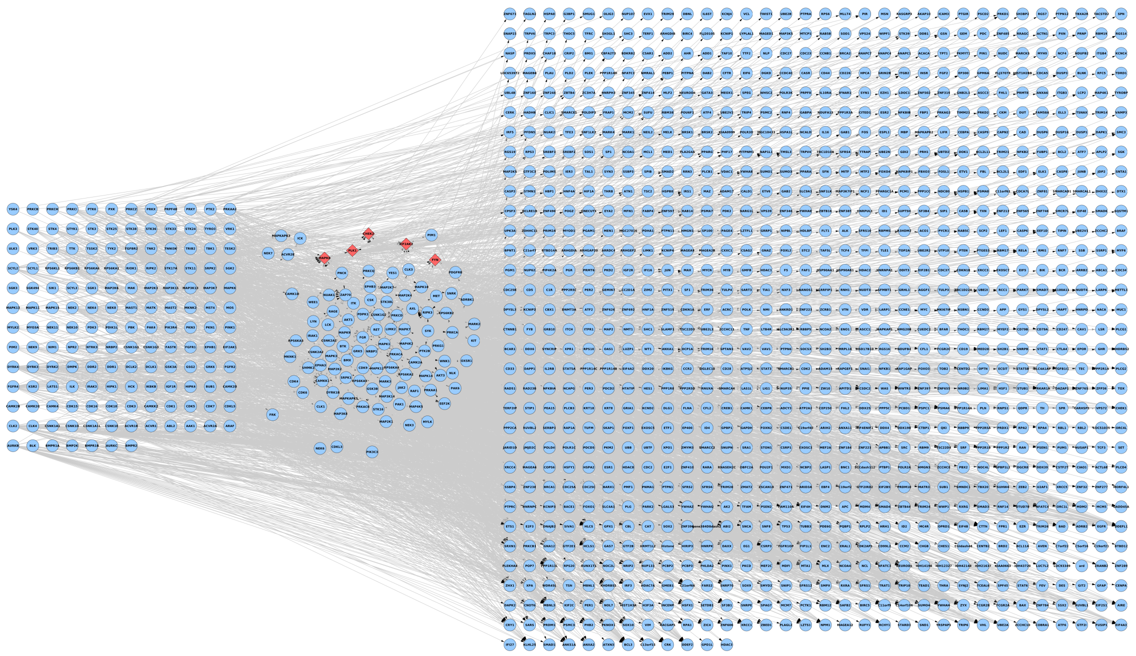

Figure 3.2: Complete view of PhosphoFull network. Edges marked red between diamond red nodes are impaired. Dissociation constants of impaired edges are increased thirty times for pathological phenotype data simulation. Left grid contains nodes that have null in-degree while right grid groups nodes with no outbound edges.

3.3.4.3 Phospho200 Mutation Network Dataset

Phospho200 Mutation network dataset use the same network structure as Phospho200 network as described in Subsection 3.3.4.2 and depicted in Figure 3.4 and Figure 3.5. The differences between Phospho2002 dataset and Phospho200 Mutation datasets are 1) Phospho200 Mutation dataset simulate gene expression profiles of patients with different single gene regulation mutations on the same pathway, while in Phospho200, all the patients have all genes on the same pathway impaired. 2) Phospho200 Mutation dataset modifies the dissociation constant of each patient's impaired edge by the factor of 10 while Phospho200 dataset modifies the dissociation constants of each patient's impaired edges by the factor of 2. Among 100 simulated patient samples, we match each 20 patient to one impaired edge in Table 3.2. This makes mutation dataset tougher for knowledge discovery but more true to the genetic changes in complex diseases like cancer.

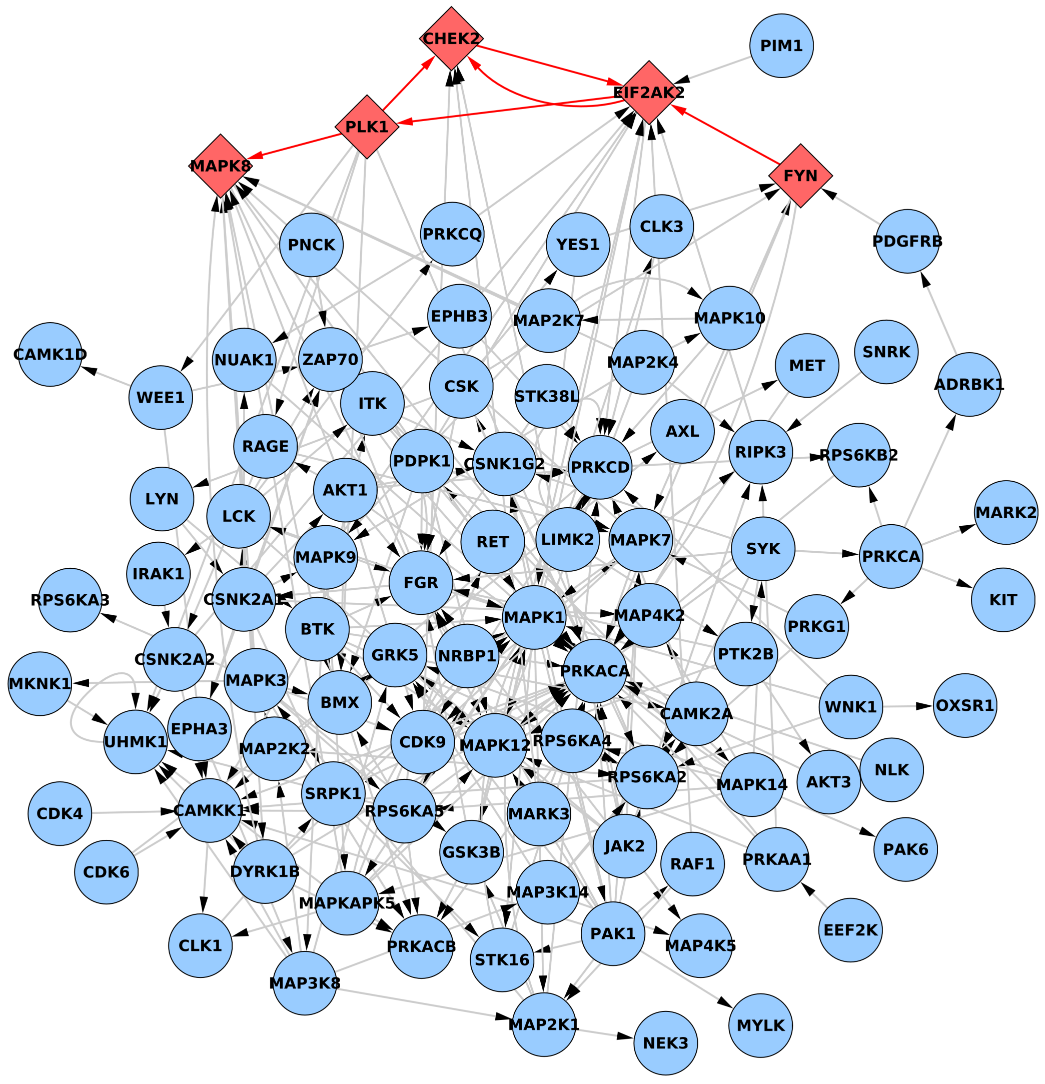

Figure 3.3: Core of the PhosphoFull network after removing nodes with null in- or out-degree. Edges marked red between diamond red nodes are impaired. Dissociation constants of impaired edges are increased thirty times for pathological phenotype data simulation.

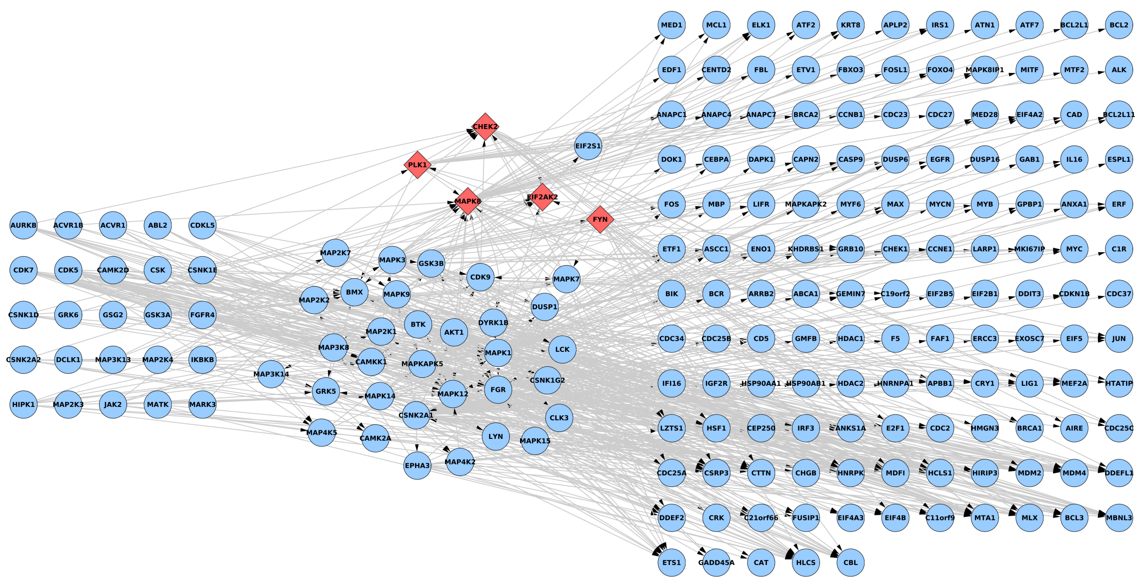

Figure 3.4: Phospho200 network extracted from full PhosphoNetwork. Edges marked red between diamond red nodes are impaired. Dissociation constants of impaired edges are increased two times for pathological phenotype data simulation. Left grid contains nodes that have null in-degree while right grid groups nodes with no outbound edges.

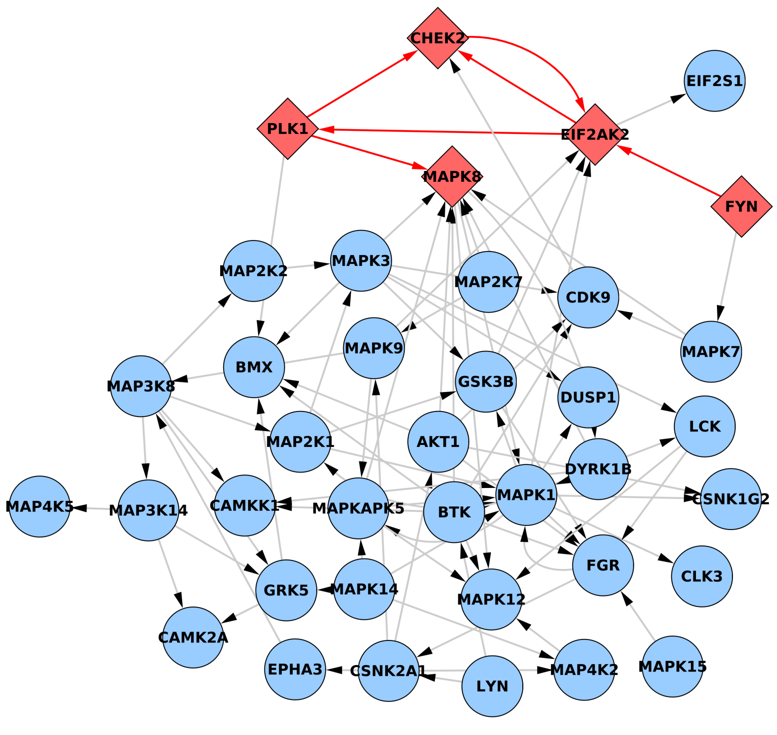

Figure 3.5: Core of the Phospho200 network after removing nodes with null in- or out-degree. Edges marked red between diamond red nodes are impaired. Dissociation constants of impaired edges are increased two times for pathological phenotype data simulation.

Chapter 4

Graph Connectivity Constrained AdaBoost

In this Chapter, we propose a new algorithm for incorporating graph information into training of the ensemble classifier AdaBoost. We will first introduce the classical AdaBoost algorithm, and then present our proposed method and the results of evaluating it on a number of datasets.

4.1 Background

4.1.1 AdaBoost Algorithm

AdaBoost is a supervised classification machine learning algorithm published by Yoav Freund and Robert Schapire in 1997 [15]. Like all ensemble methods, AdaBoost trains a set of T weak classifiers h_t, where $t = 1, ..., T$. AdaBoost name comes from "Adaptive Boosting". Adaptive means the method adjusts to the error rates ε_t of individual weak classifiers. Boosting means that weak classification method can be turned into a strong one, by aggregating many trained weak models. AdaBoost with decision tree as the base classifier is considered to be among the best out-of-the-box classifiers.

Let us begin with introducing the pseudocode of traditional Adaboost algorithm. There are two additional sets of variables in AdaBoost compared to basic ensemble scheme. Sample weights $w_t(i)$ are introduced for each sample i and for each iteration t. Weights of all samples for a given iteration form a distribution, that is, they sum up to 1. Also, weights α_t are assigned to each trained weak model h_t. In our notation, $I(a)$ returns 1 if condition a is met, and 0 if not.

Algorithm 5 AdaBoost Algorithm

```
Require: training data and classes: $(x_1, y_1), ..., (x_m, y_m)$,
  where $x_i \in \mathbb{R}^f$, $y_i \in Y = \{-1, +1\}$
Ensure: $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
  for $i \leftarrow samples\{1, ..., m\}$ do
    $w_1(i) = 1/m$;
  end for
  for $t \leftarrow iteration\{1, ..., T\}$ do
    train weak classifier $h_t$ to minimize error $\varepsilon_t = \sum_{i=1}^{m} I(h_t(x_i) \neq y_i) w_i(t)$
    assign $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\varepsilon_t}$ for classifier $h_t$
    update sample weights
        $w_{t+1}(i) = w_t(i) \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$
    normalize $w_{t+1}$ by a constant $Z_{t+1}$ so that it $\sum_{i=1}^{m} w_{t+1}(i) = 1$.
  end for
```

Essentially, AdaBoost trains a series of weak classifiers. Each weak classifier is induced, through evolving sample weights w, to learn to classify correctly the training samples that were misclassified by previously trained weak classifiers. All the trained weak classifiers form an ensemble that comes to the final decision by voting. The higher the accuracy of the weak classifier on the training set, the higher is the strength of its vote.

4.2 Proposed Method for Graph Connectivity Constrained AdaBoost

In AdaBoost, each feature is treated independently. This is not optimal if features represent genes or proteins. Previous knowledge in molecular biology reveals relationships between genes or proteins. This knowledge, though incomplete, should be taken into consideration during training of the discriminative model, since preference should be given to sets of features that represent biologically plausible groups or pathways. To incorporate biological knowledge in a form of a graph linking features, we will enhance the model to give preference to features that form connected subgraphs of the biological network.

Another limitation of AdaBoost is that once a weak classifier is chosen, it cannot be removed, nor its weight can be modified. This might introduce redundancy into the assembled classifier, especially if it is trained in an incremental way under connectivity constraints. If we want a sparse composed classifier with minimal number of necessary classifiers, the ability to flexibly delete redundant classifiers is needed for AdaBoost.

4.2.1 Connectivity-deletion AdaBoost

In this proposal, we show an approach to solving the graph-regularized ensemble learning problem. We extended AdaBoost by adding a connectivity penalty module and a deletion function. During AdaBoost classifier training iterations, we only include new weak classifiers trained on features connected in the graph to the features used by previously trained weak classifiers. Also, in the every T_d-th run, we test if by removing worst classifier or removing classifiers of the worst feature we could improve the risk function while maintaining the connectivity constrain. As we will see below, this connectivity-deletion AdaBoost shows better result on experimental datasets than traditional AdaBoost. The pseudo-code of the proposed method, along with connectivity module and two variants of the deletion function are illustrated in Algorithms 6, 7, 8, 9.

The framework of our proposed method is specified in Algorithm 6. The proposed method takes training data, sample class information and feature network information as input. Sample weights are initialized equal. They get updated and normalized by the end of each classifier generation loop. Inside the classifier generation loop, we use connectivity penalty module to select the desired new classifier. Then we apply deletion function to test if an existing classifier is detrimental to the performance of the composite classifier and can be deleted. Finally, we update sample weights using the newly defined composite classifier, which is expanded with the newly trained weak classifier, but may also be reduced by the deletion function. After T iterations, the algorithm returns the ensemble classifier H with weights α and variable importance matrix. Based on the results the proposed method provides the option of feature selection model retraining step that leads to a sparse classifier.

Compared to traditional AdaBoost (Algorithm 5 in Section 4.1.1), the proposed method

Algorithm 6 AdaCPDFC/AdaCPDTC - Connectivity-deletion AdaBoost

```
Require: : training data $X$ and classes $Y$: $(x_1, y_1), ..., (x_m, y_m)$,
  where $x_i \in \mathbb{R}^f$, $y_i \in Y = \{-1, +1\}$;
  network information of gene features $G$.
Ensure: $H(x)$ - resulting classifier.
  for $i \leftarrow samples\{1, ..., m\}$ do
    $w_1(i) = 1/m$;
  end for
  for $t \leftarrow iteration\{1, ..., T\}$ do
    $h_t = \mathbf{connectivityPenalty}(H_{t-1}, w, X, Y, G)$;
    assign $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ for classifier $h_t$;
    $H_t = \mathbf{connectivityDeletion}(H_t,\ G,\ t,\ T_d,\ cP)$;
    update sample weights: $w_i(t+1) = \exp(-y_i H_t(x_i))$;
    normalize $w_{t+1}$ so that it $\sum_{i=1}^{m} w_{t+1} = 1$;
  end for
  $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$
  optional: $H$=$\mathbf{retrain}(H, \alpha, imp, l)$
  where $imp$ is the variable importance matrix of all classifiers and all features
  and $l$ is the desired new feature size.
```

holds three novel elements: it 1) utilizes network information, 2) contains deletion function, 3) adds a retraining option. Connectivity penalty module is specified in Algorithm 7; deletion function includes two options - tree based (Algorithm 8) and feature based (Algorithm 9) deletion. The method makes use of network information in both connectivity penalty module and deletion function. The variable importance-based feature selection and model retraining option can not only improve the performance of proposed method but select the potentially impaired genes and subnetworks.

4.2.2 Connectivity Penalty Module

Connectivity penalty module (Algorithm 7) aims to select a best classifier with connected features and minimized training error. The input of module are current composite classifier H_{t-1}, sample weights w, training data and classes X, Y, and gene feature network information G. First we select a set of all the gene features that are used in the current composite classifier H_{t-1}. We call this set of features $subN$ in the module. We train decision tree $tree_{subN}$ using features from $subN$ only and calculate its training error ϵ_{subN}. Then, according to the connectivity information from G, we select all the first degree neighbor gene features of $subN$.

Algorithm 7 Connectivity Penalty Module

```
Ensure: $h_t$ = connectivityPenalty($H_{t-1}, w, X, Y, G$)
  $subN$ = all the selected features from $H_{t-1}$;
  train decision tree ($tree_{subN}$) using $subN$ with training error $\epsilon_{subN}$;
  $neibs$ = all the direct upstream and downstream neighbors of $subN$ from $G$;
  $bigN$ = union($subN$, $neibs$);
  train decision stump $stump_{bigN}$ using $bigN$;
  find the feature $f_{best}$ in $stump_{bigN}$;
  $bestN$ = all the direct upstream and downstream neighbors of $f_{best}$;
  train decision tree ($tree_{best}$) using $bestN$ with training error $\epsilon_{bestN}$;
  if $\epsilon_{bestN} < \epsilon_{subN} \cdot 0.95$ then
    return $h_t = tree_{best}$;
  else
    return $h_t = tree_{subN}$
  end if
```

We call these features *neibs* in the module. We combine features from *subN* and *neibs* to form *bigN*. We train decision stump $stump_{bigN}$ using features from *bigN* and call the feature selected in decision stump f_{best}. We then include all the first degree neighbor gene features of f_{best} together with f_{best} to form a new subnetwork, *bestN*. We then train decision tree $tree_{best}$ using features in $best_N$ and calculate its training error ε_{bestN}. Now we compare training error ε_{subN} of $tree_{subN}$ and ε_{bestN} of $tree_{bestN}$. If training error ε_{bestN} from $tree_{bestN}$ is less than 95 % of error ε_{subN} from $tree_{subN}$, we accept $tree_{bestN}$; otherwise we accept $tree_{subN}$. At this step, we have a new decision tree classifier selected. We allowed extending the set of features to form a larger connected set, but only if the classifier trained using those features is better than a classifier trained using features that were selected previously.

This module adds four properties to selected classifiers: 1) new classifier selects features only from a set that is connected in the graph; 2) all features of the composite classifier H come from a connected set in the graph; 3) no matter what feature the algorithm choose first, the connected features of the composite classifier H grow as a connected set to cover the potential impaired gene subnetwork area; 4) the composite classifier growing direction (new classifier h_t) always lowers the training error.

4.2.3 Deletion Function

Deletion function gives the method an ability to delete redundant classifiers from the composite classifier. We introduced two ways of deletion - tree based and feature based. Tree-based deletion searches for the worst weak classifier (decision tree) and removes it if this improves the performance of the composite classifier. Feature-based deletion searches for the worst gene feature and removes all classifiers that contain this feature if the removal improves the performance of the composite classifier. The connectivity module in Section 4.2.2 allows for extending the set of connected features in composite classifier to cover the "true impaired gene subnetwork" area. Our deletion function will remove "false-positive genes" from selected features and make the composite classifier "move" towards the "true impaired gene subnetwork" rather than just "grow" to incorporate it.

Both feature-based and tree-based deletion require input data that includes current composite classifier H_t after adding new classifier h_t, network information G, current iteration count t, and user defined parameters T_d and cP. The deletion happens in every T_d iterations. We take modulo of current iteration count t over deletion frequency parameter T_d and if it equals zero then we start the deletion algorithm. Parameter cP defines the stress we put on enforced connectivity - larger cP indicates smaller emphasis.

4.2.3.1 Tree-based Deletion with Enforced Connectivity

The details of implementation of Tree-based Deletion with enforced Connectivity (DTC) module are explained in Algorithm 8. First we take all the gene features f_{all} used in current composite classifier H_t. Based on gene network information, we calculate how many connected components Gc, how many features Gv are in f_{all}, and how big is the training error G_ε of H_t. Plugging in the user defined parameter cP, we can calculate the error-feature-connectivity penalty score Gp using formula $Gp = Ge + (Gv + Gc)/cP$. Then we go through each i-th weak classifier (decision tree) in the composite classifier H_t and tentatively remove that i-th classifier from H_t. In each deletion iteration we get a new composite classifier H_{-i} and we calculate its corresponding error-feature-connectivity penalty score using the same formula $GpList(i) = GeList(i) + (GvList(i) + GcList(i))/cP$. After all deletion iterations, we get

Algorithm 8 Tree-based Deletion with enforced Connectivity (DTC)

```
Ensure: $H_t$ = DTC($H_t$, $G$, $t$, $T_d$, $cP$):
  if $t \bmod T_d = 0$ then
    $f_{all}$ = all the features from $H_t$;
    $Gc$ = the number of connected components of $f_{all}$;
    $Gv$ = the number of features of $f_{all}$;
    $Ge$ = training error of $H_t$;
    the error-feature-connectivity penalty score of $H_t$: $Gp = Ge + (Gv + Gc)/cP$ ;
    for $i \leftarrow trees\{1, ..., t\}$ do
      $H_{-i}$ = delete $h_i$ from the $H_t$;
      $f_{-i}$ = features from $H_{-i}$;
      $GcList(i)$ = the number of connectivity components of $f_{-i}$;
      $GvList(i)$ = the number of features of $f_{-i}$;
      the error-feature-connectivity penalty score of $H_{-i}$:
      $GpList(i) = GeList(i) + (GvList(i) + GcList(i))/cP$;
    end for
    find the index $k$ of $Gp_{min}$ in $GpList$;
    if $Gp_{min} < Gp$ then
      return $H_t = H_{-k}$ (delete $h_k$);
    else
      return $H_t = H_t$ (no deletion);
    end if
  end if
```

the list of error-feature-connectivity penalty scores *GpList* for all classifier deletions. We take the minimum value Gp_k from *GpList* and compare with the score Gp of current composite classifier H_t. If such Gp_k is less than Gp then we accept the k-th classifier deletion from H_t and return the composite classifier H_{-k} without k-th classifier; or else, we reject deletion for this round.

4.2.3.2 Feature-based Deletion with Enforced Connectivity

Feature-based Deletion with enforced Connectivity (DFC) module presented in Algorithm 9 has a lot in common with the tree based deletion. Again, first we take all the gene features f_{all} used in current composite classifier H_t. Based on gene network information, we calculate how many connected components Gc we have in f_{all}, and how big is the training error G_ε. Using cP, we calculate the error-connectivity penalty score Gp from the formula $Gp = Ge + Gc/cP$. This penalty formula does not include feature count because we already know we only delete one feature at a time so all deletions are equal in that respect. We go through each

Algorithm 9 Feature-based Deletion with enforced Connectivity (DFC)

```
Ensure: $H_t$ = DTC($H_t$, $G$, $t$, $T_d$, $cP$):
  if $t$ mod $T_d = 0$ then
    $f_{all}$ = all the features from $H_t$;
    $Gc$ = the number of connected components of $f_{all}$;
    $Ge$ = training error of $H_t$;
    the connectivity penalty score of $H_t$: $Gp = Ge + Gc/cP$ ;
    $m$ = the number of features of $f_{all}$;
    for $i \leftarrow features\{1, ..., f\}$ do
      $H_{-i}$ = classifiers after deleting all the classifiers $\{h_j\}$ contain feature $f_i$ from the $H_t$;
      $GcList(i)$ = the number of connectivity components of $f_{-i}$, where $f_{-i}$ = delete feature
      $i$ from $f_{all}$;
      the connectivity penalty score of $H_{-i}$: $H_{-i} = GeList(i) + GcList(i)/cP$;
    end for
    find the index $k$ of $Gp_{min}$ $in$ $GpList$;
    if $Gp_{min} < Gp$ then
      return $H_t = H_{-k}$ ($delete$ $all$ $\{h_j\}$ $containing$ $f_k$).
    else
      return $H_t = H_t$ ($no$ $deletion$)
    end if
  end if
```

gene feature and tentatively remove all classifiers $\{h_j\}$ that contain the i-th gene feature from the composite classifier H_t. In each deletion iteration i we get a new composite classifier H_{-i}. We calculate its corresponding error-connectivity penalty score using the formula $GpList(i) = GeList(i) + GcList(i)/cP$. After all deletion iterations, we get the list of error-connectivity penalty scores $GpList$ for all feature deletions. We take the minimum value Gp_k from $GpList$ and compare with the score Gp of current composite classifier H_t. If such Gp_k is less than Gp then we accept the k-th feature deletion corresponding to Gp_k and return the composite classifier H_{-k} without k-th gene feature; or else, we reject deletion for this round.

4.2.4 Model Retraining

The methods outlined above aim at limiting the number and connectivity structure of the selected features. This may have detrimental effect on the accuracy of the resulting classifier. In order to improve the accuracy, while keeping only a small set of features, we introduce model retraining as a final step in our framework. Model retraining uses the top selected features, and trains a completely new classifier that uses only those features.

Algorithm 10 Variable Importance-based Retraining

```
Require: : decision trees $\{h_1, h_2, ..., h_T\}$;
  classifier weights $\alpha \leftarrow \{\alpha_1, \alpha_2, ..., \alpha_T\}$;
  variable important score matrix $imp \leftarrow \{imp_1, imp_2, ..., imp_T\}$;
  where $imp_t \leftarrow genefeaturescores\{imp_{t,1}, imp_{t,2}, ..., imp_{t,m}\}$;
  desired new feature size $l$.
Ensure: $H(x)$ - resulting classifier.
  $\overline{imp} = \sum_{t=1}^{T} imp_t \cdot \alpha_t$;
  $f_{sort} \leftarrow$ sort features according to $\overline{imp}$ in descending order;
  $f_{new} \leftarrow$ top $l$ features from $f_{sort}$;
  $H \leftarrow$ new classifier training using data that includes only $f_{new}$.
```

Because AdaBoost is not a black box model, besides training a composed classifier for prediction, we could also rank features based on their decision tree variable importance scores from weighted classifiers. **Variable importance** score is a property of decision tree. It is used to measure how important the features is for differentiating samples from different classes. Features with high variable importance score are considered to be more crucial. Features not selected by the decision tree have null variable importance.

Pseudocode for Variable Importance-based Retraining step is shown in Algorithm 10. In our proposed method, each classifier h_t has a corresponding variable importance scores imp_t for all the m features. We define feature weight score as the sum of tree-specific importance scores weighted by the tree importance in the ensemble

$$\overline{imp} = \sum_{t=1}^{T} imp_t \cdot \alpha_t \tag{4.1}$$

where α_t in 4.1 is the weight for classifier h_t. We sort all the features in a descending order according to $\overline{imp}$ and select best 5, 10, 15 and 25 features as differential gene subnetworks. To test if these selected features can improve the performance of our method by shrinking feature space, we apply either SVM or AdaBoost on data with only these selected features.

4.3 Computational Complexity

Computational complexity of traditional AdaBoost with decision tree as weak classifier is $\mathcal{O}(T(m + tree))$, where T is the number of AdaBoost iterations, m is the sample size and *tree* is the complexity of training a decision tree. Complexity of training a tree on a dataset with m samples and f features is $\mathcal{O}(f \cdot m \log m + f \cdot m \cdot h_{tree})$, where h_{tree} is the height of the trained tree. The first term corresponds to pre-sorting all samples independently along all features, and the second one to actually constructing the tree using the sorted samples. Optimistically $h_{tree} = 1$ if there is a feature that perfectly separates all the samples into two classes. Pessimistically, if at every level, the feature only separates one sample away from the rest of the samples, the tree will have $h_{tree} = m$ levels. In a balanced tree where the fraction of samples that go to one side is bounded, for example at most 90% of samples goes to one side, the tree will have $h_{tree} = \log m$ levels. So the worst case computational complexity of traditional AdaBoost with decision tree as weak classifier is $\mathcal{O}(T \cdot f \cdot m^2)$, but typically one can expect complexity of $\mathcal{O}(T \cdot f \cdot m \log m)$.

The proposed method adds a connectivity module, deletion function and retraining procedure to traditional AdaBoost framework. The retraining procedure is a traditional AdaBoost, so here we will analyze only the computational complexity of the connectivity module and the deletion function. Connectivity module is described in Algorithm 7. We assume the number selected features in H_{t-1} is V, then training $tree_{subN}$ takes $\mathcal{O}(m \log m \cdot V)$. Construction of the neighbors list may pessimistically involve a large fraction of the edges in the graph, and may take $\mathcal{O}(E)$. Training $stump_{bigN}$ takes $\mathcal{O}(m \cdot V')$ where V' is the number of features in $bigN$. Searching $bestN$ takes $\mathcal{O}(V')$ and train $treebest$ takes $\mathcal{O}(m \log m \cdot V'')$ where V'' is the number of features in $bestN$. Pessimistically, if most features in a given dataset have been used in H_{t-1}, $V \approx V' \approx V'' \approx f$ where f is the total number of features, and also the number of nodes in the graph. So the computational complexity of connectivity penalty module in total is $\mathcal{O}(f \cdot m \log m + E)$, which for a dense graph can lead to $\mathcal{O}(f \cdot m \log m + f^2)$.

Feature-based deletion function consumes more time than tree-based deletion function. Both deletion functions require a parameter T_d to define how often such deletion function is performed - every T_d iterations the deletion function will be executed. In both deletion functions, searching for the number of connected components takes $\mathcal{O}(E + f)$ where f is the number of nodes in the network equal to the number of features. This needs to be done for each tentatively removed feature, that is, f times. Also, f times, the algorithm has to inspect each of t trees, and remove those that contain the tentatively removed feature, and evaluate the performance of the ensemble classifiers of up to $t-1$ trees on m samples. So, for the single feature, quantifying the effect of its removal takes $\mathcal{O}(E+f+t+(t-1)m)$, which for f features to be analyzed adds up to $\mathcal{O}(f(E+f+tm))$, which for a dense graph can lead to $\mathcal{O}(f^3+ftm)$. Tree-based deletion function is similar to feature-based deletion function. However, instead of inspecting every gene and every classifier contains such gene, it only goes through each classifier, and the performance evaluation can be obtained by subtracting results from that one classifier from results for the ensemble, involving only $\mathcal{O}(m)$ time. The computational complexity in total is $\mathcal{O}(t(E+f+m))$, which for a dense graph becomes $\mathcal{O}(t(f^2+m))$.

Summing up all the components of our proposed method, the computational complexity of our method is pessimistically $\mathcal{O}(Tfm\log m + Tf^3 + T^2fm)$ for dense graphs, and $\mathcal{O}(Tfm\log m + Tf^2 + T^2fm)$ for sparse graphs.

4.4 Results and Discussion

We systematically assess the proposed algorithms and compare them to state-of-art methods on three datasets simulated from GeneNetWeaver as described in Chapter 3.3.2 and 3.3.3.1. In this section we present the complete performance results grouped by dataset, in Tables 4.2, 4.3 and 4.4. Methods are inspected from three aspects: 1) classification power, 2) model complexity, 3) the ability to detect the known "true differences" between phenotypes. In each table, methods are sorted by their AUROC, the gold standard statistic for model evaluation in terms of their predictive power. Model complexity is quantified as the number of features used by the trained final model. The ability to detect differences between phenotypes is measured by counting how many of the features used by the model come from two sets: the set of known "true impaired genes" and the set of first degree downstream neighboring genes of these "true impaired genes". As we explained in Chapter 3.3.4, these "true impaired genes" are directly linked to edges with modified dissociation constants in data simulation. The damaged regulation relationships influence downstream neighbor genes of the "true impaired genes" too. We present both how many known true features and first degree downstream neighbors of known true features are present in the features included in the models for each dataset.

Table 4.1: Abbreviations of Module Names

Method Type	Abbreviation	Module Name
Method frameworks	**Ada**	**Ada**Boost framework
	SVM	**S**upport **V**ector **M**achine framework
Connectivity module	**CP**	**C**onnectivity **P**enalty
Deletion functions	**DTC**	**D**eletion module, **T**ree based, enforced **C**onnectivity
	DFC	**D**eletion module, **F**eature based, enforced **C**onnectivity
Model retraining	**_Ada**	**Ada**Boost retrained using selected features
	_SVM	**S**upport **V**ector **M**achine retrained using selected features
Baseline methods	**allFeatureSVM**	**S**upport **V**ector **M**achine using **all Features**
	allFeatureAda	**Ada**Boost using **all Features**
	OracleSVM	**S**upport **V**ector **M**achine using known true impaired genes
	OracleAda	**Ada**Boost using known true impaired genes

As explained in section 4.2, the originality of our proposed method includes: 1) connectivity penalty module, 2) feature or tree based deletion function, 3) variable importance-based feature selection model retraining procedure. We test the individual and synergistic impact of these new parts of our proposed method, and compare the results of these methods to the performances of two state-of-the-art methods, AdaBoost and SVM, refered to as "Ada" and "SVM" respectively. The network connectivity constrain, connectivity penalty module is simplified as "CP" in our study. Deletion function has two versions - tree-based with enforced connectivity and feature-based with enforced connectivity. Here we call them "DTC" and "DFC", respectively. Variable importance-based model retraining procedure takes the variable importance scores for all feature as provided by the already trained ensemble, and trains a new final model using traditional AdaBoost or SVM on the top scoring features. Here we use an extension "_Ada" and "_SVM" to other methods to indicate that the model retraining procedure has been applied as the last step.

We use four configurations of state-of-the-art methods for comparison with our proposed methods. "allFeatureSVM " denotes a non-linear SVM model trained on a given dataset using all its features. "OracleSVM" denotes a non-linear SVM model trained on a given dataset using only the known true impaired gene features. "allFeatureAda" denotes traditional AdaBoost without any of our proposed extensions trained on a dataset with all features, while "OracleAda" denotes AdaBoost trained using only the known true impaired gene features. Results of the "allFeatureSVM" and "allFeatureAda" methods give us the baselines of how well these two state-of-the-art methods perform on a given dataset. “OracleSVM” and “OracleAda” give us the performance of SVM and AdaBoost, if they were equipped with an oracle that points them to the features that capture the true differences between two phenotypes, which would allow them to discard all the other features. For real datasets, such an oracle typically does not exist. The performance goal of our proposed method is to achieve accuracy as close as possible to the one obtained with the help of the oracle. The method types, abbreviations and full names are organized in Table 4.1. The details of these modules are explained in section 4.2.

All methods are implemented in MATLAB 7.10.0(R2010a). We use libsvm-3.17 MATLAB package as an SVM implementation. We use the default parameter setting from libsvm - radial basis function kernel for two-class classification SVM. We implemented our own AdaBoost function in MATLAB and used MATLAB’s toolbox for decision trees. We set the number of iterations $T = 200$ for all AdaBoost-based methods. Even though in our experiments we selected top 5, 10, 15, 20 and 25 gene features for all model retraining procedures, because the baseline “OracleAda” and “OracleSVM” functions use only five features in their models, here we only present and compare results for model retraining involving top 5 features.

4.4.1 Phospho200 Dataset

The Phospho200 dataset information is available in Chapter 3.3.4.2. Phospho200 network is extracted from PhosphoNetwork [23]. The network topology is depicted in Fig 3.4, and the nodes with non-zero both in and out degrees are magnified in Fig 3.5. The details of Phospho200 dataset simulation is explained in Chapter 3.3.2. Table 4.2 presents quantitative evaluation results of listed methods on this dataset.

We compare our results to the baseline methods “OracleAda”, “OracleSVM”, “allFeatureAda” and “allfeaturesSVM”. “OracleAda” and “OracleSVM” are two ideal situations, in which we know what are the true damaged genes in pathologic phenotype. Their results are produced by employing only these known “true features” in traditional AdaBoost and SVM algorithms. These two methods do not have any application to real life, because in real life experiments we rarely know which features are responsible for the differences between classes. Machine learning algorithms can only build models by inferring which features possibly discriminate two phenotypes. Actually, discovering the true features that divide the normal and the pathological samples is a main task of our research, so the two methods based on true features are only an unattainable hypothetical best performance of two top machine learning classification methods.

In Phospho200 dataset, “OracleAda” has the highest AUROC value 0.8706, which we can treat as the standard of the best possible performance of algorithms from the AdaBoost family. Our proposed methods “AdaCPDFC_Ada” and “AdaCPDFC_SVM” with AUROC values of 0.8079 and 0.7930 are second and third. These two configurations both are our full proposed method: a union of AdaBoost framework, connectivity penalty module, deletion function (feature-based deletion with enforced connectivity function) and variable importance-based model retraining procedure. The only difference is that AdaCPDFC_Ada uses traditional AdaBoost algorithm while AdaCPDFC_SVM uses SVM algorithm for model retraining. Behind our full proposed method comes “OracleSVM” with AUROC value 0.7893. The variations of our full proposed method with feature based deletion yield better result than the “unbeatable” ideal baseline result “OracleSVM”. Other two variations of our proposed method, with tree-based deletion, AdaCPDTC_Ada and AdaCPDTC_SVM, with AUROC values of 0.7845 and 0.7665, follow tightly behind “OracleSVM” too. This shows the success of model building of our proposed method on Phosph200 dataset. Also, the results show a preference of feature-based deletion to tree-based deletion. In feature-based deletion function, we inspect

Table 4.2: Results for Phospho200 Dataset

Method	ROC	Feature count	True features	True downstream neighbors
OracleAda	0.8706	5	5	4
AdaCPDFC_Ada	0.8079	5	1.75	3.6
AdaCPDFC_SVM	0.7930	5	1.75	3.6
OracleSVM	0.7893	5	5	4
AdaCPDTC_Ada	0.7845	5	1.7	3.5
AdaCPDTC_SVM	0.7665	5	1.7	3.5
AdaCP_Ada	0.7539	5	1.7	3.8
AdaCP_SVM	0.7536	5	1.7	3.8
Ada_Ada	0.6943	5	0.9	3.3
AdaCP	0.6770	79.3	4.95	31.2
allFeatureAda	0.6743	179.4	4.9	59.7
allFeatureSVM	0.6113	200	5	66

all genes selected by composite classifier. We go through each gene and test the training error of a new composite classifier without decision trees that contain such gene. We accept the deletion that improves the training error most, and remove all classifiers that contain the identified redundant gene feature. In tree-based deletion function, we inspect all decision trees that form the composite classifier. We tentatively remove each decision tree from the composite classifier and accept the one that lowers the training error most. In effect, only one tree can be removed at a time, and the deletion may not actually reduce the number of features used by the model.

Combination of connectivity penalty module and model retraining procedure perform well, but not as good as our full proposed method. The methods of AdaCP_Ada and AdaCP_SVM have AUROC values of 0.7539 and 0.7536. This shows the importance of deletion function. Deletion function generates small classifiers with important features. Then the AUROC values drop to 0.6943 and 0.6770 when we use model retraining procedure and connectivity module individually. Ada_Ada is the traditional AdaBoost algorithm with model retraining procedure. AdaCP is AdaBoost framework with connectivity penalty module. Traditional AdaBoost Algorithm has 0.6743 as its AUROC value while SVM using all features has 0.6113. The results show that individual usage of connectivity penalty module or model retraining procedure cannot significantly enhance the performance of AdaBoost for Phospho200 dataset.

In terms of model simplicity, “OracleAda” and “OracleSVM” employ only five features in their models because there are five known impaired gene features. To match the feature number in the ideal situation, we also selected five features in model retraining procedure. All the proposed methods that involve model retraining are set up to produce models with five features only. Traditional AdaBoost used 179.4 features on average over 10 runs of five-fold cross validations while SVM used all 200 features. So the full proposed method not only improves classification accuracy, but provides much simpler models. Furthermore, among these top five selected features, four variations of our full proposed method (AdaCPDFC_Ada, AdaCPDFC_SVM, AdaCPDTC_Ada and AdaCPDTC_SVM) manage to discover 1.75 true features and more than 3.5 true feature’s first degree downstream neighbors on average over 10 runs of five-fold cross validations.

In summary, the combination of three new modules that constitute our proposed method yields very good results - it has higher AUROC score than the ideal performance of SVM, traditional AdaBoost and SVM over all features and any performance of an individual function. Moreover, our proposed method generates models with only a few features that drastically shrink the number of features traditional AdaBoost and SVM use. The proposed method is accurate, sparse and, although not perfectly, recovers the true sources of changes.

We can also observe that methods that use AdaBoost show consistently better results than methods that use SVM: "allFeatureAda" has greater AUROC than "allfeatureSVM", "OracleAda" has greater AUROC than "OracleSVM" and model retrain procedure favors "_Ada over "_SVM" too. This might be because AdaBoost does not require parameter tuning, and we are only using default kernel and parameters for SVM model. Full parameter study may improve the results for SVM, but our results can be view as the comparison of the methods when they are treated as out-of-the-box classifiers.

4.4.2 Phospho200 Mutation Dataset

Phospho200 Mutation dataset is a harder classification problem than Phospho200 dataset. The Phospho200 Mutation data is designed to simulate patients with single gene mutation and single gene pair deregulation along a disease pathway. Among 100 pathologic samples, every 20 samples share one unique gene mutations. These 5 mutations together define a disease pathway. We increased the dissociation constants of damaged gene regulations 10 times. In Phopho200 dataset, all 100 pathologic samples have all 5 mutations, and we increase dissociation constants of the damaged gene relations 2 times. So similar to a real biological situation, there are greater but less uniform gene expression alterations in the mutation dataset. The Phospho200 Mutation dataset information is available in Chapter 3.3.4.3. Phospho200 Mutation dataset and Phospho200 network use the same network structure, which is extracted from PhosphoNetwork [23]. The network topology is depicted in Figures 3.4 and 3.5. The details of data set simulation is explained in Chapter 3.3.2.

Table 4.3 presents quantitative evaluation results of listed methods on Phospho200 Mutation dataset. Two variations of our full proposed method, AdaCPDFC_Ada and AdaCPDFC_SVM have high AUROC values of 0.8079 and 0.7930. The results are close to the ideal result "OracleSVM" with AUROC value 0.8134. This confirms the advantage

Table 4.3: Results for Phospho200 Mutation Dataset

Method	ROC	Feature count	True features	True downstream neighbors
OracleAda	0.8680	5	5	4
OracleSVM	0.8134	5	5	4
AdaCPDFC_Ada	0.8079	5	1.75	3.6
AdaCP_Ada	0.8078	5	1.9	3.7
AdaCP_SVM	0.8058	5	1.9	3.7
AdaCPDFC_SVM	0.7930	5	1.75	3.6
AdaCPDTC_Ada	0.7845	5	1.7	3.5
AdaCPDTC_SVM	0.7665	5	1.7	3.5
allFeatureAda	0.7181	181.75	4.95	60.35
AdaCP	0.7174	76.65	4.8	31.4
Ada_Ada	0.7141	5	1	3.55
allFeatureSVM	0.6434	200	5	66

of our full proposed method in classification problems. The full proposed method with tree based deletion function has good AUROC values of 0.7845 and 0.7665 too. However, in mutation dataset, excluding the tree based deletion procedure actually enhances the method performance. AdaCP_Ada and AdaCP_SVM obtain higher values of AUROC values of 0.8078 and 0.8058, which is an even better results than AdaCPDFC_SVM, AdaCPDTC_Ada and AdaCPDTC_SVM. From the results of Phosph200 dataset and Phosph200 mutation dataset, we can see AdaCPDFC_Ada has a consistent best performance among all practical methods. We also find a consistent order of four variations of our full proposed method: AdaCPDFC_Ada, AdaCPDFC_SVM, AdaCPDTC_Ada and AdaCPDTC_SVM, indicating feature-based deletion outperforms tree-based deletion. Connectivity penalty module and model retraining procedure alone again fail to show better results than traditional AdaBoost, but they still performs much better than SVM with default parameter over all features.

In terms of model complexity, we get similar result to Phopho200 dataset. All full proposed methods build models with only five features, among which 1.7 to 1.9 features are true features and 3.5 to 3.7 features are first downstream neighbors of true features. All the presented results are averaged over 10 runs of 5 fold cross validation.

To sum up, in both Phospho200 and Phospho200 Mutation dataset: 1) proposed method yields good results that are competitive with “OracleSVM”, although there is still room for improvement compared to “OracleAda”; 2) the full proposed method AdaCPDFC_Ada works best; 3) out of the box, with default parameters, AdaBoost-based methods perform better than corresponding methods based on SVM; 4) combination of connectivity, deletion and retraining performs best in Phopho200 dataset, while the combination of connectivity and retraining show some advantages in mutation dataset; 5) all proposed methods largely improve the performance of traditional AdaBoost algorithm.

4.4.3 PhosphoFull Dataset

PhosphoFull dataset has 200 samples and 1291 features. It is much bigger dataset than Phospho200 and Phosph200 Mutation dataset. It contains the same five impaired relationships as Phospho200. For those impaired relationships, we increased the dissociation constants 30 times in PhosphoFull dataset, for all 100 pathologic samples. The PhosphoFull dataset information is available in Chapter 3.3.4.1.

Table 4.4 presents quantitative evaluation results of listed methods on PhosphoFull dataset. In this dataset, three variations of our full proposed method obtained AUROC scores only slightly below "OracleSVM" and "OracleAda". The order of the four variations of our proposed method changed compared to previous datasets, but again feature-based deletion outperforms tree-based deletion. AdaCPDFC_SVM performs best with AUROC value of 0.7882, followed by 0.7747 from AdaCPDTC_SVM and 0.7653 from AdaCPDFC_Ada. Then we have the methods combine connectivity penalty module and model retraining pro-

Table 4.4: Results for PhosphoFull Dataset

Method	ROC	Feature count	True features	True downstream neighbors
OracleAda	0.8025	5	5	4
OracleSVM	0.7974	5	5	4
AdaCPDFC_SVM	0.7882	5	1.55	3.2
AdaCPDTC_SVM	0.7747	5	1.6	2.95
AdaCPDFC_Ada	0.7653	5	1.55	3.2
AdaCP_SVM	0.7587	5	1.6	2.85
allFeatureAda	0.7585	437.5	3.55	68.1
AdaCP_Ada	0.7582	5	1.6	2.85
Ada_Ada	0.7472	5	1.25	3.3
AdaCPDTC_Ada	0.7456	5	1.6	2.95
AdaCP	0.7071	118.05	3.5	35.45
allFeatureSVM	0.5425	1291	5	148

cedure AdaCP_SVM with 0.7587. Though AdaCP_Ada, Ada_Ada, AdaCPDTC_Ada and AdaCP fail to beat the AUROC value of traditional AdaBoost “allFeatureAda”, they manage to surpass “allFeatureSVM”. Furthermore, all methods with model retraining procedure include only 5 features in their models. Compared to 437.5 features in traditional AdaBoost model, our methods provide much smaller classifiers with improved AUROC results. Our full proposed methods recovered 1.55 to 1.6 true features and 2.95 to 3.2 first degree neighbors of true features among 5 features they select. This is similar discovery rate to Phospho200 and Phosph200 mutation dataset.

In a nutshell, our proposed Graph Connectivity Constrained AdaBoost method is able to produce classifiers with high accuracy, small number of features and discover part of the true genetic difference between phenotypes in larger dataset.

Chapter 5

Linear Support Vector Machine with Submodular Graph Regularization

In this Chapter, we propose a new method for graph regularization in linear Support Vector Machines (SVM), by introducing a graph penalty that is based on the theory of submodular set functions. We will first review the SVM and Linear Programming SVM algorithms, as well as the relevant background about submodular set functions, and then present our proposed method Graph Regularized Linear SVM and the results of its evaluation.

5.1 Background

5.1.1 Linear Programming Support Vector Machine

Linear Support Vector Machines aim at searching for an optimal decision hyperplane to maximize sample margin. Because we can geometrically prove the norm of feature weights is inversely proportional to margin $M = \frac{2}{\|\beta\|}$, we can solve the classification problem by looking for a set of feature weights β that minimizes the squared L_2 norm of β and the total sample risk expressed as the sum of slack variables ξ

$$\text{minimize} \quad \frac{1}{2}\|\beta\|_2^2 + C\sum_{i=1}^{m}\xi_i \tag{5.1}$$

$$\text{subject to} \;\; \xi_i \geq 0, \;\; y_i(x_i^T\beta + b) \geq 1 - \xi_i. \tag{5.2}$$

By y_i we represent the class of sample i, and the feature values of sample i are represented by the vector x_i.

The objective function in 5.1 is comprised of two terms. The first term $\frac{1}{2}\|\beta\|_2^2$ is deducted from geometry of SVM. It can also be understood as a regularization term of feature importance. The second term $C\sum\limits_{i=1}^{m}\xi_i$ bounds from above the number of misclassified samples. Because of the constrain $\xi_i \geq 1 - y_i(x_i^T\beta + b)$, total sample risk can also be written in the form of $C\sum\limits_{i=1}^{m}\mathcal{L}((y_i(x_i^T\beta + b)))$, where $\mathcal{L}$ is the hinge loss of the classifier that represents how much error the model makes. The second term guides SVM to select support vectors from samples.

The first term in the objective function in 5.1 is in the form of L_2-norm of feature weights. Even though L_2 is the traditional from of SVM feature regularization, it is not the only possible form. Several authors [26,62] proposed SVMs that use the L_1-norm of feature weights, $\|\beta\|_1$, to replace the L_2-norm term $\frac{1}{2}\|\beta\|_2^2$. This pushes more feature weights to zero and leads to a sparser SVM model as a result. Because $\|\beta\|_1 = \sum\limits_{j=1}^{F}|\beta_j|$, L_1 feature weight regularization of linear soft margin SVM can be presented as

$$\text{minimize} \quad \sum_{j=1}^{F}|\beta_j| + C\sum_{i=1}^{m}\xi_i \tag{5.3}$$

$$\text{subject to} \;\; \xi_i \geq 0, \;\; y_i(x_i^T\beta + b) \geq 1 - \xi_i. \tag{5.4}$$

This version of SVM can be solved by Linear Programming and is often referred to as Linear Programming SVM, or LP SVM.

Hybrid approaches have also been proposed before. The Doubly Regularized SVM is the application of elastic net regularization, that is, for L_2 and L_1 norm in SVM [53]. Zou [63] proposed a hybrid linear SVM to improve the performance of LP SVM. In the objective function of L_1 feature regularization of SVM, Zou adjust the L_1-norm penalty term using the feature weights from the L_2-norm SVM result. Hybrid SVM is a way to achieve better classification accuracy and feature selection ability than traditional L_1-norm and L_2-norm of SVM.

5.1.2 Submodular Functions and their Convex Extensions

In our graph regularization, we will be adding a penalty based on a submodular set function defined over the set of selected features. To that end, below we review basic results from the theory of submodular set functions.

A real-valued set function Ω defined on the power set 2^V of a set $V = 1, ..., F$ is submodular [1] if and only if

$$\forall A, B, \{a\} \subseteq V,\ A \subseteq B,\ \Omega(A \cup \{a\}) - \Omega(A) \geq \Omega(B \cup \{a\}) - \Omega(B). \tag{5.5}$$

This equation demonstrates the diminishing return property of submodular set functions. That is, as the size of the input set of submodular function grows, the increase of output value by adding the same element $\{a\}$ decreases.

To give some intuition, consider equipping every element j in the set V with a fixed weight w_j. First, let us define a set function $\Omega_1(\mathcal{S}) = \sum_{j \in \mathcal{S}} w_j$. For this set function, the weak inequality in eq. (5.5) always holds with equality. $\Omega_1(\mathcal{S})$ is an example of a modular function.

Now consider another set function $\Omega_2(\mathcal{S}) = \max_{j \in \mathcal{S}} w_j$. Adding an element may increase the maximum of w over the set, but the increase we see by adding it to A is greater than or equal to the increase we see by adding it to B, which contains all the elements from A, and some more, perhaps with higher weights. Thus, eq. (5.5) always holds, the function Ω_2 is submodular.

Finally, consider a set function $\Omega_3(\mathcal{S}) = -\max_{j \in \mathcal{S}} w_j$. For this function, the reverse of inequality in eq. (5.5) is always true, and that function Ω_3 is said to be supermodular. For every submodular function, its negation is supermodular.

A modular function is both submodular and supermodular. One important example is the set cardinality function, that is, the $\Omega_1(\mathcal{S})$ defined above for the case where all $w_j = 1$.

That function is modular, and thus also submodular.

One well-known submodular function is graph cut capacity, defined for a graph G with vertices V and edge weights G_{jk}. The graph cut function Ω_G for a selected subset of vertices $\mathcal{S} \subset V$ is the sum of edge weights between vertices in $\mathcal{S}$ and all other vertices in V (see Figs. 5.1 and 5.2). For undirected graphs, graph cut function can be formulated in the following way

$$\Omega_G(\mathcal{S}) = \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} G_{jk}, \tag{5.6}$$

while for directed graphs, graph cut function can be written as

$$\Omega_G(\mathcal{S}) = \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} G_{jk} + \sum_{j \in \mathcal{S}} \sum_{k \notin \mathcal{S}} G_{kj}. \tag{5.7}$$

We can prove that eq. (5.6) represents a submodular set function, by showing that it meets the condition as expressed in eq. (5.5). We will use $G(A \leftrightarrow B)$ to represent the sum of weights of edges linking nodes in set A and nodes in set B. For every set $A,\ B \subseteq V$ with $A \subseteq B$ and every $x \in V \setminus B$, proving $\Omega_G(A \cup \{x\}) - \Omega_G(A) \geq \Omega_G(B \cup \{x\}) - \Omega_G(B)$ is thus

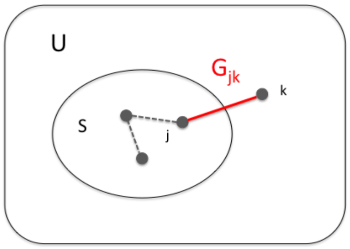

Figure 5.1: Graph cut regularizer penalizes undirected edges that cross set $\mathcal{S}$ boundary.

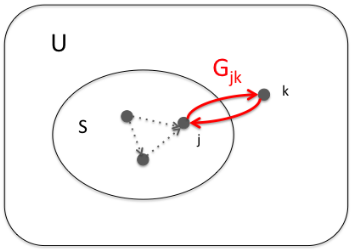

Figure 5.2: Graph cut regularizer penalizes directed edges that cross set $\mathcal{S}$ boundary.

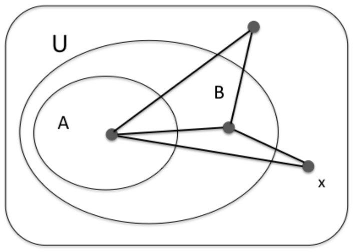

Figure 5.3: By adding element x, a smaller set A receives bigger increase of edge numbers that cross the expanded set boundary than a larger set B. U represents the universe of set elements, equal to the set of graph vertices V.

equivalent to proving that

$$G((A\cup\{x\}) \leftrightarrow (V\backslash(A\cup\{x\}))) - G(A \leftrightarrow (V\backslash A)) \geq G((B\cup\{x\}) \leftrightarrow (V\backslash(B\cup\{x\}))) - G(B \leftrightarrow (V\backslash B)). \tag{5.8}$$

Looking at the left-hand side, we can see that

$$G((A\cup\{x\}) \leftrightarrow (V\setminus(A\cup\{x\}))) - G(A \leftrightarrow (V\setminus A)) = G(\{x\} \leftrightarrow (V\setminus(A\cup\{x\}))) - G(A \leftrightarrow \{x\}). \tag{5.9}$$

Similarly, on the right-hand side we have

$$G((B\cup\{x\}) \leftrightarrow (V\setminus(B\cup\{x\}))) - G(B \leftrightarrow (U\setminus B)) = G(\{x\} \leftrightarrow (V\setminus(B\cup\{x\}))) - G(B \leftrightarrow \{x\}). \tag{5.10}$$

Because $A \subseteq B$, we know that $G(\{x\} \leftrightarrow (V \setminus (A \cup \{x\}))) \geq G(\{x\} \leftrightarrow (U \setminus (B \cup \{x\})))$ and that $G(A \leftrightarrow \{x\}) \leq G(B \leftrightarrow \{x\})$, which immediately lead to the proof that graph cut is submodular. The configuration involving A, B, and x is depicted in Fig. 5.3.

One interesting property of submodular set function is their relationship with convex and concave functions. Notably, for every submodular set function defined over subsets of set V with cardinality F, we can construct its Lovasz extension onto unit hypercube in $\mathbb{R}^F$. Every dimension corresponds to one element in the set V, and for a set $\mathcal{S} \subset V$, we can denote by $\mathrm{supp}(\mathcal{S})$ a vector in $[0,1]^F$ that represents the support of the set $\mathcal{S}$, that is, a vector with 1 at coordinates corresponding to elements from V that are in $\mathcal{S}$, and 0 at other coordinates. In that way, $\mathrm{supp}(\mathcal{S})$ corresponds to a corner of the unit hypercube $[0,1]^F$.

Lovasz extension of a submodular set function $\Omega : 2^V \to \mathbb{R}$ is equivalent to the set

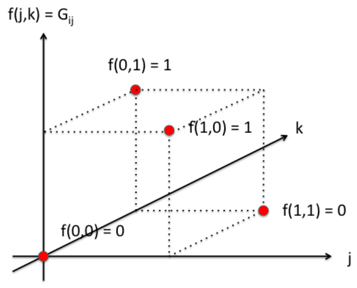

Figure 5.4: Graph cut set function and its values for a single specific edge are defined at points (0,0), (0,1), (1,0), (1,1). We assume $G_{jk} = 1$ in the plot.

function's convex closure [13], defined as a function $\Omega^L : [0,1]^F \to \mathbb{R}$ that is the point-wise highest convex function from $[0,1]^F$ to $\mathbb{R}$ such that $\Omega^L(\text{supp}(\mathcal{S})) = \Omega(\mathcal{S})$. From convexity and from equality of the submodular set function to its Lovasz extensions at the corners of the unit hypercube, we can conclude that the minimum of the extension is equal to the minimum of the set function, and is attained at one of the corners of the unit hypercube. Often, the practice is to extend the function beyond the unit hypercube, while preserving the above characteristics. For example, for the set cardinality function $\Omega_L(\mathcal{S}) = \sum_{j\in\mathcal{S}} 1$, the Lovasz extension is

$$\Omega_L^L(\beta) = \sum_{j=1}^{F} |\beta_j| \,, \tag{5.11}$$

which is known in the machine learning community as the LASSO regularization term.

For the graph cut submodular set function $\Omega_G(\mathcal{S}) = \sum_{j\in\mathcal{S}} \sum_{k\notin\mathcal{S}} G_{jk}$ (eq. 5.6) defined over a graph with F vertices and edge weights G_{jk}, the Lovasz extension is

$$\Omega_G^L(\beta) = \sum_{j=1}^{F} \sum_{k=1}^{F} G_{jk} \, |\beta_j - \beta_k| \,. \tag{5.12}$$

This can be seen by observing that the function can be decomposed into terms corresponding to individual pairs j and k connected by an edge, and analyzing the behavior of Ω_G^L for β_j and β_k. For specific j and k, the graph cut function Ω_G in eq. (5.6) has values only at four points - $(0,0)$, $(1,0)$ $(0,1)$ $(1,1)$. Such function is illustrated in Fig. 5.4. When nodes j and k are both inside or both outside the selected subset $\mathcal{S}$ of nodes, we have $\Omega_G(0,0) = \Omega_G(1,1) = 0$. When node j is inside $\mathcal{S}$ and node k is outside $\mathcal{S}$, or node k is inside $\mathcal{S}$ and node j is outside $\mathcal{S}$, we have $\Omega_G(1,0) = \Omega_G(0,1) = 1$.

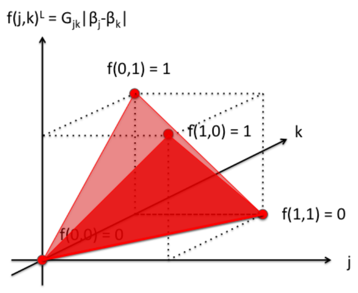

Figure 5.5: Lovasz extention of the graph cut submodular function is convex, piece-wise linear, and equal to the original submodular set function at the corners of the unit hypercube, (0,0), (0,1), (1,0), (1,1). We assume $G_{jk} = 1$ in the plot.

Since Ω_G^L has to be highest point-wise convex function, we see that $\Omega_G^L = 0$ everywhere on the line connecting (0,0) with (1,1). By the same argument, it has to be linear on both sides of that line. The function in eq. (5.12) is the only one that meets those requirements. Any point-wise higher function will no longer be convex. The function is depicted in Fig. 5.5.

5.2 Proposed Method for Submodular Graph-based Regularization in Linear SVM

5.2.1 Graph-based Regularization in SVM

In this Chapter, we propose graph regularized LP SVM. This new classification algorithm lets prior knowledge of feature relationships play a role in model training. It summarizes feature relationships as a graph regularization penalty term and integrates such term into SVM loss function. The graph regularization term derived from feature network G can provide us with another perspective on feature importance for feature selection: features that are neighbors of a very important feature are likely to be more important themselves. The graph regularization term drives the algorithm towards selecting clusters of related features, instead of a collection of independent features. The connected features may yield better classification results and more interpretable models.

One way of adding a term that promotes the selection of connected subsets of features is to penalize for every edge (in the feature graph G) that connects a feature selected by the model with a feature that is not selected. The submodular graph cut function $\Omega_G(\mathcal{S})$, where

$\mathcal{S}$ represents the set of features with non-zero coefficients β_j in the SVM model represents that type of penalty. While graph cut function Ω_G is a set function and we cannot directly add it to the objective function being optimized by SVM, we can have an equivalent function defined over the corners of the unit hypercube in $\mathbb{R}^F$, that is, over the support of feature coefficients vector β

$$\Omega_G(\text{supp}(\beta)) = \sum_{j=1}^{F} \sum_{k=1}^{F} G_{jk} \left|\text{supp}(\beta_j) - \text{supp}(\beta_k)\right|, \tag{5.13}$$

where

$$\text{supp}(\beta_j) = \begin{cases} 0, & \text{if } \beta_j = 0, \\ 1, & \text{if } \beta_j \neq 0. \end{cases} \tag{5.14}$$

While feature weights indicate the feature importance in a linear SVM model, the support of feature weights vector indicates if features are selected in the model or not. The 0/1 values from support of β divide features into two groups, forming a cut in the feature network graph. Graph cut function defined in eq. (5.13) goes through each edge of feature network and counts how many edges the cut crosses. Given the same graph G and the same number of features being selected, to obtain a smaller cut we have to minimize $|\text{supp}(\beta_j) - \text{supp}(\beta_k)|$ when $G_{jk} \neq 0$. In a graph, this means we want adjacent nodes connected by an edge to be the same group, either both selected or both not selected. So, if we add such graph cut function to SVM objective function, it will drive our models to favor the selection of features that cluster together. However, the graph cut function is non-convex and non-linear. Adding such a term into SVM objective function will make the problem untractable.

To solve this problem, instead of adding $\Omega_G(\text{supp}(\beta))$ we can use its Lovasz extension Ω_G^L, which is defined of the vector of feature weights β, and is convex, although not differentiable. Thus, we use the Lovasz extension of graph cut submodular set function as the final graph regularization term and add it to the LP SVM objective function, and we obtain our proposed

Graph Regularized Linear SVM

$$\text{minimize} \quad C\sum_{i=1}^{m}\xi_i + \sum_{j=1}^{F}|\beta_j| + C'\sum_{j=1}^{F}\sum_{k=1}^{F}|\beta_j - \beta_k|\,G_{jk} \tag{5.15}$$

$$\text{subject to} \quad \xi_i \geq 0,\ y_i(x_i^T\beta + b) \geq 1 - \xi_i, \tag{5.16}$$

In order to solve linear programming problem with absolute values, we introduce new variables β_j^+ and β_j^- for each variable β_j. As long as the coefficient in front of β_j is non-negative in the objective function, which is the case in our linear program, we will obtain positive feature weights β_j^+ and the absolute value of negative feature weights β_j^- so that $|\beta_j| = \beta_j^+ + \beta_j^-$ and $\beta_j = \beta_j^+ - \beta_j^-$. We also let $d_{jk} = \beta_j - \beta_k$ so that $|\beta_j - \beta_k| = |d_{jk}|$ and introduce new variables d_{jk}^+ and d_{jk}^- for each variable d_{jk}. In this way $|d_{jk}| = d_{jk}^+ + d_{jk}^-$ and $d_{jk} = d_{jk}^+ - d_{jk}^-$ is true.

From $\beta_j - \beta_k = d_{jk}$, $d_{jk} = d_{jk}^+ - d_{jk}^-$, $\beta_j = \beta_j^+ - \beta_j^-$, $\beta_k = \beta_k^+ - \beta_k^-$, we derive another constraint $d_{jk}^+ - d_{jk}^- - \beta_j^+ + \beta_j^- + \beta_k^+ - \beta_j^- = 0$. Now the optimization defining our proposed Graph Regularized Linear SVM can be readily seen as a linear programming problem

$$\text{minimize} \quad C\sum_{i=1}^{m}\xi_i + \sum_{j=1}^{F}(\beta_j^+ + \beta_j^-) + C'\sum_{j=1}^{F}\sum_{k=1}^{F}(d_{jk}^+ + d_{jk}^-)G_{jk} \tag{5.17}$$

$$\text{subject to} \quad \xi_i,\ \beta_j^+,\ \beta_j^-,\ d_{jk}^+,\ d_{jk}^- \geq 0, \tag{5.18}$$

$$y_i(x_i^T\beta + b) \geq 1 - \xi_i, \tag{5.19}$$

$$d_{jk}^+ - d_{jk}^- - \beta_j^+ + \beta_j^- + \beta_k^+ - \beta_j^- = 0. \tag{5.20}$$

The problem can be solved efficiently using standard techniques.

5.3 Results and Discussion

5.3.1 Biological Datasets

We compare our Graph Regularized Linear SVM (GLPSVM) method results with LP SVM and classical linear SVM with L_1 norm over slack variables (L1SVM). We list the area under ROC curve value and model feature number for each method in the Table 5.1.

Table 5.1: Linear SVM results for Phospho200 Dataset

Method	AUROC	Feature count
GLPSVM	0.8600	5
LPSVM	0.8455	16
L1SVM	0.7470	99

Table 5.2: Linear SVM results for PhosphoFull Dataset

Method	AUROC	Feature count
GLPSVM	0.8015	20
LPSVM	0.7895	10
L1SVM	0.6330	644

Graph Regularized Linear SVM has higher AUROC value and lower selected feature number than LP SVM and L1 SVM. We can say graph regularization improves the performance of LP SVM by increasing prediction accuracy and reducing model size in Phosph200 dataset.

PhosphoFull dataset results are listed in Table 5.2. Graph Regularized Linear SVM obtains higher AUROC value than LP SVM and L1 SVM. However, LP SVM selected smaller number of features than Graph Regularized Linear SVM.

Phosph200 mutation dataset results are listed in Table 5.3. Graph Regularized Linear SVM outperforms LP SVM and L1 SVM by its higher AUROC value and smaller model size.

In summary, in biological dataset Phospho200, PhosphoFull and Phospho200 Mutation, Graph Regularized Linear SVM performs best among all methods. Graph regularized SVM consistently achieves higher ROC values than LP VM and L1 SVM. It also controls the size of selected features small. From the results, we can conclude that Graph Regularized Linear SVM performs well on biological datasets.

Table 5.3: Linear SVM results for Phospho200 Mutation Dataset

Method	AUROC	Feature count
L1SVM	0.6667	93
GLPSVM	0.6542	29
LPSVM	0.6347	28

Table 5.4: Linear SVM results for Gaussian Red Dataset

Method	AUROC	Feature count
GLPSVM	0.9250	11
LPSVM	0.9130	23
L1SVM	0.8000	56

Table 5.5: Linear SVM results for Gaussian Cyan Dataset

Method	AUROC	Feature count
GLPSVM	0.9090	45
LPSVM	0.9020	22
L1SVM	0.7550	55

5.3.2 Benchmark Non-biological Datasets

We also compared results of Graph Regularized Linear SVM with LP SVM and L1 SVM on three simulated Gaussian datasets. The results are presented in Tables 5.4 5.5 5.6. Graph Regularized Linear SVM and LP SVM perform consistently better than L1 SVM in all three Gaussian datasets. Graph Regularized Linear SVM and LP SVM obtain higher AUROC values than L1 SVM and select fewer features for models. In Red and Cyan datasets, Graph Regularized Linear SVM achieves higher AUROC values than LP SVM. In Red dataset, such difference is greater and Graph Regularized Linear SVM selects smaller number of features. In Cyan dataset, such difference is vague and GLPSVM selects larger number of features. In Green dataset, Graph Regularized Linear SVM has lower AUROC value than LP SVM. However, Graph Regularized Linear SVM selects less features. From results of Gaussian datasets, we can conclude that, graph-regularization works better if the differential subnetwork is more internally connected and less connected to other nodes that are not in the subnetwork.

We also compared the proposed Graph Regularized Linear SVM with LP SVM and L1 SVM on time series dataset. Results in Table 5.7 shows GLPSVM obtains higher ROC than LP SVM and L1 SVM. However, GLPSVM selects largest number of features among all three

Table 5.6: Linear SVM results for Gaussian Green Dataset

Method	AUROC	Feature count
LPSVM	0.9160	29
GLPSVM	0.9040	18
L1SVM	0.8105	54

Table 5.7: Linear SVM results for time series Dataset

Method	AUROC	Feature count
GLPSVM	0.7381	449
LPSVM	0.7240	25
L1SVM	0.6981	390

Table 5.8: Linear SVM results for MNIST Dataset

Method	Error Rate [%]	Feature count
GLPSVM	5.29	696
L1SVM	5.41	536
LPSVM	5.62	160

methods. Finally, we wanted to test the effect of graph regularization in image analysis so we applied the same set of three SVM methods to MNIST handwriting digits dataset. Results are presented in Table 5.8. It should be noted that the Error Rate and not the AUROC is the community standard for presenting the results on this dataset, since the sizes of the classes are equal. From the table we can tell that the proposed Graph Regularized Linear SVM reduced the prediction error rate of LP SVM. However, it selects more features than LPSVM.

Chapter 6

Boosting with Proximal Descent and Submodular Graph Regularization

In this Chapter, we propose a new method for graph regularization in boosting, by introducing a graph penalty that is based on the theory of submodular set functions. We will first review the theory of boosting viewed as a descent method. We will also provide introduction the the proximal gradient technique that will be essential for our proposed graph-based regularization in boosting. Then, we will present our proposed method ProxGraphLogisticBoost and the results of its evaluation.

6.1 Background

6.1.1 AdaBoost as a Descent Method in Functional Space

The AdaBoost ensemble learning algorithm 5 reviewed in Chapter 4.1.1 can be seen as a method for minimizing a risk functional through the coordinate-wise steepest descent. AdaBoost and other modern classification algorithms are more powerful than classical methods because they aim at maximizing the margin rather than minimizing error of classifiers [5, 12, 16, 34]. For a binary classifier H, the margin of a sample (x_i, y_i) is defined as

$y_iH(x_i)$. The task of a good classifier is to maximize the margins of all training samples. However, not every sample weights the same - adjusting the decision boundary to move samples from the wrong side of the decision boundary to the right side is much more important than moving samples that are already on the right side of decision boundary and far away from it even farther. A loss function $\mathcal{L} : \mathbb{R} \to \mathbb{R}$ of sample margins is needed to represent such non-uniformity of importance. In AdaBoost, exponential function of negated sample margin is used as loss function. AdaBoost aims to find a model H that minimizes the average value of the loss function of all sample margins in the training set, also referred to as the empirical risk $\mathcal{R}[H]$ of the hypothesis H. Given the training set $S = \{(x_1, y_1), ..., (x_m, y_m)\}$, where $x_i \in \mathbb{R}^N, y_i \in Y = \{-1, +1\}$, the empirical risk is

$$\mathcal{R}[H] = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(-y_iH(x_i)). \tag{6.1}$$

Empirical risk is a functional $\mathcal{R} : \mathcal{F} \to \mathbb{R}$ defined over a space $\mathcal{F}$ of decision functions taking values in [-1,1].

Following the optimization explanation of boosting [34], we can start with defining an inner product $< f, g >$ for $f, g \in \mathcal{F}$ as their similarity over the training set

$$< f, g >= \frac{1}{m}\sum_{i=1}^{m} f(x_i)g(x_i). \tag{6.2}$$

For a functional $\mathcal{R} : \mathcal{F} \to \mathbb{R}$, we define its gradient as the functional $\nabla\mathcal{R}$ such that

$$\lim_{\alpha\to 0}\frac{\mathcal{R}[F+\alpha f] - \mathcal{R}[F] - < \nabla\mathcal{R}[F], \alpha f >}{\alpha} = 0. \tag{6.3}$$

Plugging in the definitions of the risk (6.1) and the inner product (6.2), we obtain a limit that links gradient of the risk functional with the loss function

$$\lim_{\alpha\to 0}\frac{\frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(-y_i(F(x_i)+\alpha f(x_i)) - \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(-y_i(F(x_i)) - \frac{1}{m}\sum_{i=1}^{m}\nabla\mathcal{R}[F](x_i)\alpha f(x_i)}{\alpha} = 0. \tag{6.4}$$

The limit converges if all terms related to individual samples x_i inside the sum converge. Since $f(x_i)$ is a real number for a single sample x_i, we can treat it as a multiplicative constant in front of α which does not affect convergence to zero

$$\lim_{(\alpha f(x_i))\to 0} \frac{\mathcal{L}(-y_i(F(x_i)+(\alpha f(x_i)))) - \mathcal{L}(-y_i(F(x_i)) - \nabla\mathcal{R}[F](x_i)(\alpha f(x_i))}{(\alpha f(x_i))} = 0. \tag{6.5}$$

For an individual sample x_i, we can treat $F(x_i)$ and $\alpha f(x_i)$ as real variables, with $F(x_i) = \hat{y}_i$ and $\alpha f(x_i) = \Delta\hat{y}_i$, leading to

$$\lim_{\Delta\hat{y}_i\to 0} \frac{\mathcal{L}(-y_i(\hat{y}_i+\Delta\hat{y}_i)) - \mathcal{L}(-y_i\hat{y}_i) - \nabla\mathcal{R}[F](x_i)\Delta\hat{y}_i}{\Delta\hat{y}_i} = 0, \tag{6.6}$$

which shows that $\nabla\mathcal{R}[F](x_i)$ meets the condition of the classical derivative of $\mathcal{L}(-y_i\hat{y}_i)$ interpreted as a function of $\hat{y}_i$

$$\begin{aligned}\nabla\mathcal{R}[F](x_i) &= \left.\frac{\partial\mathcal{L}(-y_i\hat{y}_i)}{\partial\hat{y}_i}\right|_{\hat{y}_i=F(x_i)} \\ &= \left.\frac{\partial\mathcal{L}(z_i)}{\partial z_i}\right|_{z_i=-y_iF(x_i)} \left.\frac{\partial z_i}{\partial\hat{y}_i}\right|_{\hat{y}_i=F(x_i)},\end{aligned} \tag{6.7}$$

where we have introduced $z_i = -y_iF(x_i)$. Thus we obtain the final representation of the gradient of the risk functional $\mathcal{R}$ applied to function F and evaluated at x_i

$$\nabla\mathcal{R}[F](x_i) = -y_i \left.\frac{\partial\mathcal{L}(z_i)}{\partial z_i}\right|_{z_i=-y_iF(x_i)}. \tag{6.8}$$

In AdaBoost, we build the final ensemble in an iterative fashion. At the start of iteration t, we have a fixed ensemble model H_{t-1} and we are aiming to expand it to $H_t = H_{t-1} + \alpha_t h_t$. Thus, we need to find the new classifier h_t so that the value of $\mathcal{R}[H_t]$ is lower than $\mathcal{R}[H_{t-1}]$. Specifically, our goal in iteration t is to find a decision function h_t and a parameter α_t that result in the smallest risk

$$(h_t, \alpha_t) = \arg\min_{h,\alpha} \mathcal{R}[H_{t-1} + \alpha h]. \tag{6.9}$$

The direction the risk decreases most rapidly is simply the negative of the gradient of risk at H_{t-1}, that is, $-\nabla\mathcal{R}[H_{t-1}]$. Ideally, we would set the new h_t to be the direction along which our risk functional decrease most rapidly from its current value of H_{t-1} and the new α_t to how far we are going to move in that direction. That is, we would like to choose $h_t = -\nabla\mathcal{R}[H_{t-1}]$. But we may not always find such classifier in the space $\mathcal{H}$ of models returned by our particular weak classification algorithm, so in practice we choose a classifier h_t that maximizes the inner product $\langle -\nabla\mathcal{R}[H_{t-1}]), h_t\rangle$. We terminate the algorithm when $\langle \nabla\mathcal{R}[H_{t-1}], h_t\rangle = 0$ which means when the h_t can no longer point to the downhill direction of risk functional $\mathcal{R}[H]$.

From the definition of the inner product (6.2) and the final form of the gradient of H (6.8), we can express the criterion to be maximized for the new weak classifier h_t as

$$-\langle \nabla\mathcal{R}[H_{t-1}], h_t\rangle = \frac{1}{m}\sum_{i=1}^{m} h(x_i) y_i \left.\frac{\partial \mathcal{L}(z_i)}{\partial z_i}\right|_{z_i=-y_i H_{t-1}(x_i)}. \tag{6.10}$$

We can treat the derivative as a weight of a sample x_i at iteration t, that is, after H_{t-1} is fixed

$$w_i(t) = \frac{1}{Z_t} \left.\frac{\partial \mathcal{L}(z_i)}{\partial z_i}\right|_{z_i=-y_i H_{t-1}(x_i)}, \tag{6.11}$$

where Z_t is a normalization constant making $w_i(t)$ a distribution

$$Z_t = \sum_{i=1}^{m} \left.\frac{\partial \mathcal{L}(z_i)}{\partial z_i}\right|_{z_i=-y_i H_{t-1}(x_i)}. \tag{6.12}$$

Plugging the weights into the inner product, we obtain the criterion to be maximized as

$$-\langle \nabla\mathcal{R}[H_{t-1}], h_t\rangle = \frac{Z_t}{m}\sum_{i=1}^{m} y_i w_i(t) h_t(x_i) = \frac{Z_t}{m}\gamma_t, \tag{6.13}$$

where $\gamma_t \in [-1, 1]$ is often called the edge of a classifier and is related to the weighted error $\varepsilon = \frac{1-\gamma}{2}$. All variables except h_t are fixed at the start of iteration t, so as long as a new classifier h_t minimizes the weighted error of the classifier, and thus maximizes the weighted edge on the training set, it also maximizes the inner product of itself with $-\nabla\mathcal{R}[H_{t-1}]$. That means it is the classifier close to the gradient direction of current risk function.

For the exponential loss function $\mathcal{L}(z_i) = \exp(z_i)$, we recover the weights as presented in Algorithm 5

$$-\langle \nabla\mathcal{R}[H_{t-1}], h_t\rangle = \frac{1}{m}\sum_{i=1}^{m} y_i \exp(-y_i H_{t-1}(x_i)) h(x_i) = \frac{Z_t}{m}\sum_{i=1}^{m} y_i w_i(t) h(x_i), \tag{6.14}$$

where

$$w_i(t) = \frac{1}{Z_t} \exp(-y_i H_{t-1}(x_i)). \tag{6.15}$$

We can observe that the weights for the next iteration, if we ignore the normalization constants, are

$$w_i(t+1) = \exp(-y_i H_t(x_i)) = \exp(-y_i H_{t-1}(x_i) - y_i \alpha_t h_t(x_i)) = w_i(t) \exp(-y_i \alpha_t h_t(x_i)), \tag{6.16}$$

which is equivalent to the iterative update used in the AdaBoost algorithm if we observe that $y_i h_t(x_i)$ is either 1 or -1, depending on whether x_i is classified correctly by h_t. We see that gradient descent defines how AdaBoost sets its sample weight distribution and how it sets the optimization goal for training the new weak classifier. Since the exponential loss function used in AdaBoost is convex, gradient descent will not be trapped in a local minimum.

Once the weak classifier h_t for iteration t is trained, we can interpret the empirical risk for the exponential loss as just a real-valued function of a single real variable α and find the optimal value of α by tools from real analysis. After dropping the multiplicative constant $\frac{Z_t}{m}$ for convenience, we can define $\mathcal{R}(\alpha_t) : \mathbb{R} \to \mathbb{R}$ as

$$\mathcal{R}(\alpha_t) = \sum_{i=1}^{m} \exp(-y_i H_{t-1}(x_i) - y_i \alpha_t h_t(x_i)) = \sum_{i=1}^{m} w_i(t) \exp(-y_i h_t(x_i) \alpha_t), \tag{6.17}$$

where everything except α_t is constant by the time we set to choose α_t. We then partition the sum into terms relating to misclassified and correctly classified samples, obtaining

$$\begin{aligned}
\mathcal{R}(\alpha_t) &= e^{-\alpha_t} \sum_{i:h_t(x_i)=y_i} w_i(t) + e^{\alpha_t} \sum_{i:h_t(x_i)\neq y_i} w_i(t) \\
&= e^{-\alpha_t} \left(\sum_{i=1}^{m} w_i(t) - \sum_{i:h_t(x_i)\neq y_i} w_i(t) \right) + e^{\alpha_t} \sum_{i:h_t(x_i)\neq y_i} w_i(t) \\
&= e^{-\alpha_t} + (e^{\alpha_t} - e^{-\alpha_t}) \sum_{i:h_t(x_i)\neq y_i} w_i(t) \\
&= e^{-\alpha_t} + (e^{\alpha_t} - e^{-\alpha_t}) \sum_{i=1}^{m} I(h_t(x_i) \neq y_i) w_i(t) \\
&= e^{-\alpha_t} + (e^{\alpha_t} - e^{-\alpha_t}) \varepsilon_t.
\end{aligned} \tag{6.18}$$

To find the minimum, we take derivative

$$\frac{\partial \mathcal{R}(\alpha_t)}{\partial \alpha_t} = -e^{-\alpha_t} + (e^{\alpha_t} + e^{-\alpha_t})\varepsilon_t. \tag{6.19}$$

Setting the derivative equal to zero we obtain

$$0 = -1 + (e^{2\alpha_t} + 1)\varepsilon_t. \tag{6.20}$$

which leads to

$$\frac{1-\varepsilon_t}{\varepsilon_t} = e^{2\alpha_t} \tag{6.21}$$

from where we obtain

$$\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t}. \tag{6.22}$$

This is exactly how AdaBoost defines weight of each classifier weight in Algorithm 5. If classifier h_t makes less errors on weighted samples then it will get larger weight α_t in the ensemble.

6.1.2 Boosting as a Descent Method in the Space of Real Vectors

For a specific training set of m samples, the space of possible weak classifiers $\mathcal{H}$ is finite as long as the weak learning algorithm is deterministic. There are 2^m possible binary classification outcomes from a classifier on the training set, and thus we have only 2^m effectively distinct weak classifiers. Furthermore, if we use decision stumps as weak classifiers, for a particular dataset with F features and m samples, the number of possible stumps with different outcomes is limited by the number of features F, times the number of possible effectively different splits $m+1$, times two possible orientations of a split. That is, the limit on the number of different decision stumps is $2F \times (m+1)$, which is typically much less than 2^m.

With that in mind, it is often conceptually more convenient to work in the space $\mathbb{R}^{2^m}$ than in the space of functions h, where the 2^m-dimensional vector ϕ represents the weight of each of the possible weak classifiers in the ensemble. The process of building the ensemble starts with $\phi = 0$, that is with no weak classifier selected, and then moves one individual component

ϕ_{i_1} away from zero when the first weak learner returns a particular weak classifier, by adding the weight α_1 to entry ϕ_{i_1} of the vector ϕ, where i_1 is the classifier that was selected by the weak learning algorithm. In general, in round t of boosting, component ϕ_{i_t} is incremented, by adding the weak classifier weight α_t. After t weak classifiers are added to the ensemble, only up to t components of ϕ are nonzero. It may be less that t if a specific weak classifier is selected more than once by the weak learning algorithm, although typically this is unlikely to happen because the weights of the samples are constantly evolving.

From this perspective, boosting is a process of minimizing a risk function $\mathcal{R} : \mathbb{R}^{2^m} \to \mathbb{R}$ defined as

$$\mathcal{R}(\phi) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(-y_i \sum_{j=1}^{2^m} \phi_j h_{ij}), \tag{6.23}$$

where $h_{ij} = h_j(x_i)$, the label for sample x_i predicted by the weak classifier h_j corresponding to the dimension ϕ_j, can be treated as a real constant. That constant is unknown to us until the classifier h_j is returned by the weak learning algorithm, but before that happens the coefficient ϕ_j remains 0, so the value of the constant is not needed.

The partial derivative of $\mathcal{R}$ with respect to ϕ_k evaluated at ϕ is then

$$\frac{\partial \mathcal{R}}{\partial \phi_k}(\phi) = \frac{1}{m} \sum_{i=1}^{m} -y_i h_{ik} \mathcal{L}'(-y_i \sum_{j=1}^{2^m} \phi_j h_{ij}), \tag{6.24}$$

where the term $\mathcal{L}'(-y_i \sum_{j=1}^{2^m} \phi_j h_{ij})$ can be seen as a weight of the sample i corresponding to current ensemble ϕ. As already noted, most of the terms ϕ_j in the sum are zero and need not be considered, so evaluating the weights is efficient. Then, finding the axis ϕ_k with the most negative $\frac{\partial \mathcal{R}}{\partial \phi_k}$ corresponds to minimizing the weighted error of the weak classifier.

In summary, we can view one step in boosting as a coordinate-wise descent, that is

$$\phi^{t+1} = \phi^t - s^t \nabla_{\max} \mathcal{R}(\phi^t), \tag{6.25}$$

where s^t is the step size, and ϕ^t and ϕ^{t+1} are non-negative vectors of weak classifier weights. We have introduced $\nabla_{\max}\mathcal{R}$ to represent the risk gradient vector $\nabla\mathcal{R}$ modified to include only the most negative $\frac{\partial \mathcal{R}}{\partial \phi_k}(\phi)$, with all other vector entries set to 0.

Plugging in the exponential loss $\mathcal{L}(x) = e^x$ with $\mathcal{L}'(x) = e^x$ into eq. (6.25) leads to classical AdaBoost. Other choices have also been explored, for example for logistic loss $\mathcal{L}(x) = \log(1 + e^x)$ we obtain $\mathcal{L}'(x) = \frac{e^x}{1+e^x}$, which leads to different formula for weights of samples in boosting, and may result in a different choice of weak classifiers.

6.1.3 Proximal Gradient Descent

We have seen that AdaBoost, as well boosting with other loss functions such at the logistic loss, can be seen as a descent method for optimizing a convex risk function. Here, we review techniques from non-smooth function optimization [3, 41] that will help us add graph-based regularization into boosting. First, it is helpful to cast gradient descent as a special case of majorization-minimization (MM) scheme [55].

If we cannot directly find the minimum of function $f(\phi)$, the MM approach prescribes an iterative method that goes through a series of solutions $\phi^1, \phi^2, ..., \phi^t, ...$ until convergence. For convex functions, convergence to within a small error away from the global minimum is guaranteed. The MM approach constructs the series $\phi^1, \phi^2, ..., \phi^t, ...$ through an introduction of a family of majorizing functions $\mu_{\phi^1}(\phi), \mu_{\phi^2}(\phi), ..., \mu_{\phi^t}(\phi), ...$, where the majorizer $\mu_{\phi^t}(\phi)$ is defined based on the information about the function f we have at the time t when we are at solution ϕ^t. These majorizers have the following properties:

$$\forall \phi \;\; \mu_{\phi^t}(\phi) \geq f(\phi), \tag{6.26}$$

$$\mu_{\phi^t}(\phi^t) = f(\phi^t). \tag{6.27}$$

In iteration t we are at position ϕ^t, and the majorizing function μ_{ϕ^t} is greater than or equal to f at any ϕ, and is equal to f at the current position, ϕ^t. The MM approach is helpful if we can find a majorizing function that can easily be optimized. For example, if μ is a quadratic function, we can find its minimum efficiently. Then, we can easily obtain next point in the series,

$$\phi^{t+1} = \arg\min_{\phi} \mu_{\phi^t}(\phi), \tag{6.28}$$

and proceed towards the minimum of f. That descent happens because

$$f(\phi^{t+1}) \leq \mu_{\phi^t}(\phi^{t+1}) \leq \mu_{\phi^t}(\phi^t) = f(\phi^t). \tag{6.29}$$

As long as we are not already at the minimum of either f, and the shape of μ_{ϕ^t} meets certain weak conditions, the second weak inequality will turn to strong inequality. In defining the function $\mu_{\phi^t}(\phi)$ at iteration t, we can use all the information about the function f we have discovered up to that point. This may include values of f and gradients of f at all visited solutions up to and including ϕ_t.

For a convex function f with Lipschitz continuous gradient with constant L, that is, for f that meets the following criterion

$$\forall \phi, \psi \quad ||\nabla f(\phi) - \nabla f(\psi)|| \leq L||\phi - \psi||, \tag{6.30}$$

it is well know that the following descent lemma is true for any ϕ, ϕ^t

$$\begin{aligned} f(\phi) &\leq f(\phi^t) + < \nabla f(\phi^t), \phi - \phi^t > + \frac{L}{2}||\phi - \phi^t||^2 \\ &= f(\phi^t) + \frac{L}{2}||\phi - [\phi^t - \nabla f(\phi^t)/L]||^2 - \frac{1}{2L}||\nabla f(\phi^t)||^2. \end{aligned} \tag{6.31}$$

Thus, we can construct the following quadratic majorizing function

$$\mu_{\phi^t}(\phi) = f(\phi^t) + \frac{L}{2}||\phi - [\phi^t - \nabla f(\phi^t)/L]||^2 - \frac{1}{2L}||\nabla f(\phi^t)||^2. \tag{6.32}$$

From the descent lemma we will get that $f(\phi) \leq \mu_{\phi^t}(\phi)$, and it can be easily seen that $f(\phi^t) = \mu_{\phi^t}(\phi^t)$, so $\mu_{\phi^t}(\phi)$ is a majorizer of f at ϕ^t. Only the second term depends on ϕ, so we can treat the other terms as a constant C_{ϕ^t} depending on ϕ^t but not ϕ, and thus playing no role in the minimization of $\mu_{\phi^t}(\phi)$

$$\mu_{\phi^t}(\phi) = C_{\phi^t} + \frac{L}{2}||\phi - [\phi^t - \nabla f(\phi^t)/L]||^2. \tag{6.33}$$

We can see that $\mu_{\phi^t}(\phi)$ is a quadratic function of ϕ with minimal value when

$\phi = \phi^t - \nabla f(\phi^t)/L$. That is, the MM approach leads to a step $\phi^{t+1} = \phi^t - \nabla f(\phi^t)/L$, a single step of classical gradient descent.

The benefit of viewing gradient descent as an example of the MM approach becomes clear once we move to minimizing a composite function $f(\phi) + \Omega(\phi)$, where both f and Ω are convex, but Ω is not differentiable, so we cannot take its gradient to perform regular gradient descent.

If $\mu_{\phi^t}(\phi)$ majorizes f, then $\mu_{\phi^t}(\phi) + \Omega(\phi)$ majorizes $f(\phi) + \Omega(\phi)$, and we have

$$\mu^{\Omega}_{\phi^t}(\phi) = \mu_{\phi^t}(\phi) + \Omega(\phi) = C_{\phi^t} + \frac{L}{2}||\phi - [\phi^t - \nabla f(\phi^t)/L]||^2 + \Omega(\phi). \tag{6.34}$$

The step from ϕ^t to ϕ^{t+1} in the MM process becomes then

$$\phi^{t+1} = \arg\min_{\phi} \mu^{\Omega}_{\phi^t}(\phi) = \arg\min_{\phi} \left(\frac{L}{2}||\phi - [\phi^t - \nabla f(\phi^t)/L]||^2 + \Omega(\phi) \right), \tag{6.35}$$

and by virtue of the properties of the MM scheme, we obtain convergence to the global minimum of $f(\phi) + \Omega(\phi)$, as long as both f and Ω are convex.

It is convenient to define the proximal operator that takes a vector ψ as input, returns a vector ϕ from the same vector space, and is parameterized by a real-valued parameter L and a parameter function Ω defined over the same vector space

$$\text{prox}_{L,\Omega}(\psi) = \arg\min_{\phi} \left(\frac{L}{2}||\phi - \psi||^2 + \Omega(\psi) \right). \tag{6.36}$$

This definition leads to the following concise formulation of the iterative minimization through proximal gradient descent

$$\psi^{t+1} = \phi^t - \frac{1}{L}\nabla f(\phi^t), \tag{6.37}$$

$$\phi^{t+1} = \text{prox}_{L,\Omega}(\psi^{t+1}). \tag{6.38}$$

The step from ϕ^t to ϕ^{t+1} involves a gradient step followed by a proximal step. Most importantly, as long as the proximal operator can be solved efficiently, the minimization process does not involve the gradient of Ω, which means Ω does not need to be differentiable.

In practice, the constant L may be hard to estimate, and more importantly may be large, since it has to account for the shape of the function f in its whole domain. Instead of using $\frac{1}{L}$ as the gradient step, we can use an adjustable real-valued step size s^t

$$\psi^{t+1} = \phi^t - s^t \nabla f(\phi^t), \tag{6.39}$$

$$\phi^{t+1} = \mathrm{prox}_{\frac{1}{2s^t},\Omega}(\psi^{t+1}). \tag{6.40}$$

Step size s^t can be determined using line search and will correspond better to the curvature of the objective function in the vicinity of the current solution ϕ^t than the generic step size of $\frac{1}{L}$, and thus will result in faster convergence.

6.2 Proposed Method for Submodular Graph Regularization in Boosting

Classical boosting is an iterative method for minimizing the convex empirical risk $\mathcal{R}(\phi)$ without any constraint or regularization. To incorporate biological knowledge in a form of a graph linking features, we may want to add additional terms in the objective function optimized during model training. Such terms will be derived from the network of feature relations and will be added to boosting risk function and optimized together with sample error. Thus, we change the problem into minimizing $\mathcal{R}(\phi) + \Omega(\phi)$ where Ω represents a penalty for the structure of the model, and may be based on graph G. One option would be to use a submodular set function over the support of ϕ. However, that would make the problem not convex and not tractable.

To circumvent that problem, we could follow the approach we presented for linear SVMs - using the convex, piece-wise linear Lovasz extension of a submodular set function, for example a graph cut function. In addition, we could add a LASSO penalty, since originally boosting does not have any terms that would keep the number of selected features from growing as the size of the ensemble grows. The resulting objective function to be minimized would be

$$\mathcal{R}(\phi) + \Omega_G^L(\phi) + \Omega_L^L(\phi), \tag{6.41}$$

where Ω_G^L represents the Lovasz extension of the graph cut submodular function as defined in eq. (5.12), and Ω_L^L is the LASSO penalty, that is, the Lovasz extension of the modular cardinality function (see eq. 5.11) . Both of those regularizing functions are convex, but non-differentiable. On the other hand $\mathcal{R}$ is convex and differentiable. Thus, it seems natural to apply the techniques of proximal optimization described above in Section 6.1.3 to find the global minimum of the regularized objective function.

There are several issues that need to by solved before we can move forward with that approach. First, the regularization terms should be defined in terms of the F-dimensional vector of feature importance scores β, not in terms of a much larger space of possible weak classifier weights ϕ. The graph G has F vertices corresponding to F features, not to all weak classifiers. Also, with the LASSO penalty, we want to limit the number of used features, not the number of weak classifiers, which is already limited by the number of boosting rounds.

This problem can be dealt with by observing that each weak classifier is sparse, and uses a limited number of features, in the case of decision stumps just one feature. We can define a mapping matrix Γ with each column corresponding to a possible weak classifier, and each row corresponding to one feature. For decision stumps, each column would have only a single 1, in the row corresponding to the feature being used by the stump. The rest of the column would be 0. Then, we can obtain feature importances at time t from the weights of weak classifiers at time t by a simple linear transformation $\beta^t = \Gamma\phi^t$. The transformation will group together weak classifiers that use a particular feature, add their weights together, and use that as the importance score for the feature.

The matrix Γ is fully specified by the choice of weak learner and the size of the training set, but individual columns become known only after the corresponding weak classifier is trained. That is, column i becomes known when classifier weight ϕ_i becomes non-zero. Fortunately, only at that time the content of the column affects the vector β, and needs to be known. With this approach, the minimization problem becomes

$$\mathcal{R}(\phi) + \Omega_G^L(\Gamma\phi) + \Omega_L^L(\Gamma\phi). \tag{6.42}$$

Since the transformation we introduced is linear, the objective function is still convex. In practice, we can treat the new problem as a convex problem with linear constraints

$$\text{minimize} \quad \mathcal{R}(\phi) + \Omega^L_G(\beta) + \Omega^L_L(\beta) \tag{6.43}$$

$$\text{subject to} \;\; \beta = \Gamma\phi. \tag{6.44}$$

Second problem with applying the proximal technique is that the exponential loss function used in AdaBoost does not have Lipschitz continuous gradient. Exponential function is arbitrarily steep if we get far away from zero. However, we can switch to the logistic loss $\mathcal{L}(x) = \log(1 + e^x)$ which has Lipschitz continuous gradient. Logistic loss has previously been used in boosting with good results.

Third problem with applying techniques from proximal optimization to regularized boosting is related to the fact that boosting is a coordinate-wise descent, not gradient descent. That is, instead of the direction of $-\nabla\mathcal{R}$, the step is in the direction of $-\nabla_{\max}\mathcal{R}$. That is, boosting moves along only one axis, the one with the steepest decline in the value of the objective function. Even if a function f has Lipschitz continuous gradient ∇f with some constant L (see eq. 6.30), we may have a situation when

$$\forall K \; \exists \phi, \psi : \quad ||\nabla_{\max} f(\phi) - \nabla_{\max} f(\psi)|| > K||\phi - \psi||. \tag{6.45}$$

One example is a two-dimensional function $f(x_1, x_2) = x_1^2 + x_2^2$, which has Lipschitz continuous gradient with $L = 2$. For $\phi = (K, K-1)$ and $\psi = (K-1, K)$ we have $||\phi - \psi|| = \sqrt{2}$, $||\nabla f(\phi) - \nabla f(\psi)|| = 2\sqrt{2}$, but $||\nabla_{\max} f(\phi) - \nabla_{\max} f(\psi)|| = 2K\sqrt{2}$. That is, a small change to the input ϕ, from $(K, K-1)$ to $(K-1, K)$, can result in arbitrarily large change to $\nabla_{\max} f(\phi)$. In consequence, the descent lemma (eq. 6.31) does not hold, and the function $\mu^{\Omega}_{\phi^t}(\phi)$ constructed in the way described above in Section 6.1.3 may not be a proper majorizer of the objective function, which means convergence to the minimum is not guaranteed.

There are two reasons why $\nabla_{\max}\mathcal{R}$ is used in boosting instead of $\nabla\mathcal{R}$. First, boosting was envisioned as an iterative ensemble where the classifiers are added one by one, and the weights of the previously added classifiers stay untouched. There is no fundamental reason preventing us from changing the weights of these weak classifiers by evaluating the partial derivatives corresponding to them. To this end, we can define ∇_{t+1} as a vector

$$\nabla_{t+1}\mathcal{R} = \left[\frac{\partial \mathcal{R}}{\partial \phi_1}, \frac{\partial \mathcal{R}}{\partial \phi_2}, .., \frac{\partial \mathcal{R}}{\partial \phi_t}, \frac{\partial \mathcal{R}}{\partial \phi_{t+1}}, 0, 0, ...\right], \tag{6.46}$$

and use it instead of $\nabla_{\max}$ in each boosting iteration. In this way, in step t, we will add one new weak classifier, corresponding to dimension ϕ_{t+1}, and we will also adjust weights for the previously selected weak classifiers. Note that for simplicity, in defining ∇_{t+1} we assumed, without loss of generality, that the ordering of classifiers in the space of all weak classifiers is such that classifier number $t+1$ is the one that is being returned by the weak learner and added to the ensemble when we move from solution ϕ^t to solution ϕ^{t+1}. The full ordering is unknown to us and is dependent on the algorithm and training set we use, and is revealed to us one weak classifier per one round of boosting.

The other reason for not using $\nabla\mathcal{R}$ in boosting is more fundamental. Going back to the definition of $\frac{\partial \mathcal{R}}{\partial \phi_j}$ (see eq. 6.24), we see that to calculate it, we need to evaluate the term $\mathcal{L}'(-y_i \sum_{j=1}^{2^m} \phi_j h_{ij})$, which depends on knowing the training set predictions h_{ij} of the weak classifier corresponding to axis ϕ_j. We know these only for the weak classifiers that are already in the ensemble. In a sense, only those classifiers are real, they are a result of running the weak learning algorith, for example a decision stump. All other weak classifiers from our chosen weak classifier space $\mathcal{H}$ are potentially available, but we don't know what they are yet. Thus, while it is possible to calculate $\nabla_{t+1}\mathcal{R}$ instead of $\nabla_{\max}\mathcal{R}$, calculating $\nabla\mathcal{R}$ is impractical, since we would have to train and evaluate all possible weak classifiers that can result from our choice of the weak learner, for example all possible decision stumps.

One thing that is realistic to assume, though, is that we can obtain h_{ij} for an additional small set of weak classifiers in each round of boosting. Thus, in place of $\nabla_{t+1}\mathcal{R}$ we can define $\nabla_{\mathcal{T}}\mathcal{R}$, where $\mathcal{T}$ is a set of weak classifiers for which we either already know the predictions h_{ij} from previous boosting rounds, or we are willing to spend some computational time to obtain the predictions. In the simplest case of regular boosting, in each round we train just one new weak classifier, and then $\mathcal{T}$ contains only the first $t+1$ weak classifiers, and $\nabla_{\mathcal{T}}\mathcal{R}$ reduces to $\nabla_{t+1}\mathcal{R}$. But we have the possibility to go as much beyond that as computationally feasible. All that remains to be shown is that we will be able to come up with a majorization-minimization scheme that does not require evaluating full gradient $\nabla\mathcal{R}$ to guarantee convergence to global minimum.

Here, we show that by replacing the majorizing function $\mu^{\Omega}_{\phi^t}(\phi)$ (eq. 6.34) with a different majorizer, proximal gradient descent using $\nabla_{\mathcal{T}}\mathcal{R}$ is equivalent to proximal gradient descent using $\nabla\mathcal{R}$. Thus, since the minimization scheme is valid for $\nabla\mathcal{R}$, it will be valid for $\nabla_{\mathcal{T}}\mathcal{R}$. Specifically, let us define a new majorizer

$$\tau^{\Omega}_{\phi^t}(\phi) = \mu^{\Omega+\infty_{\mathcal{T}}}_{\phi^t}(\phi) = \mu^{\Omega}_{\phi^t}(\phi) + \infty_{\mathcal{T}}(\phi), \tag{6.47}$$

where

$$\infty_{\mathcal{T}}(\phi) = \begin{cases} \infty, & \text{if } \exists j \notin \mathcal{T} \; : \; \phi_j > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{6.48}$$

As long as non-zero entries in ϕ correspond to classifiers that we already trained, that is, those from the set $\mathcal{T}$, the value of $\infty_{\mathcal{T}}(\phi)$ is null. If we have a non-zero entry anywhere else, $\infty_{\mathcal{T}}(\phi^t)$ is infinite. We can see that $\mu^{\Omega}_{\phi^t}(\phi) \leq \tau^{\Omega}_{\phi^t}(\phi)$, with equality for $\phi = \phi^t$. Thus, since $\mu^{\Omega}_{\phi^t}(\phi)$ is a majorizer of $\mathcal{R}(\phi) + \Omega(\phi)$, so is $\tau^{\Omega}_{\phi^t}(\phi)$ (see definition of a majorizing function in eq. 6.26). We thus obtain a proper majorization-minimization scheme that will first make a gradient step, and then will next make a proximal step with the updated parameter function, $\Omega + \infty_{\mathcal{T}}$ instead of just Ω.

The benefit of introducing a new majorizer is that the minimum of $\tau^{\Omega}_{\phi^t}$ is limited to vectors with zeros everywhere except possibly the entries listed in the set $\mathcal{T}$. For every other vector, $\tau^{\Omega}_{\phi^t} = \infty$, so they cannot be the minimum. As a consequence, even if we perform the gradient step using full gradient $\nabla\mathcal{R}$, any movement in the direction outside of the subspace spanned by the axes from $\mathcal{T}$ will be canceled by the proximal step. Thus, performing the gradient step using $\nabla_{\mathcal{T}}\mathcal{R}$ instead of $\nabla\mathcal{R}$ will not impact the outcome of the majorization-minimization procedure involving the $\tau^{\Omega}_{\phi^t}$ majorizer.

The final procedure for iterative minimization of $\mathcal{R}(\phi) + \Omega(\phi)$ is then

$$\psi^{t+1} = \phi^t - s^t \nabla_{\mathcal{T}} \mathcal{R}(\phi^t), \tag{6.49}$$

$$\phi^{t+1} = \operatorname{prox}_{\frac{1}{2s^t}, \Omega + \infty_{\mathcal{T}}}(\psi^{t+1}), \tag{6.50}$$

where s^t is obtained using line search. The proximal step in round t corresponds to the following constrained quadratic problem

$$\text{minimize} \quad \frac{1}{2s^t} ||\phi - [\phi^t - s^t \nabla_{\mathcal{T}} \mathcal{R}(\phi^t)]||^2 + \Omega(\beta) \tag{6.51}$$

$$\text{subject to } \beta = \Gamma\phi, \tag{6.52}$$

$$\phi_j = 0 \;\; \forall j \notin \mathcal{T}. \tag{6.53}$$

In practice, we can drop the last constraint involving ϕ_j and perform optimization over a shortened vector ϕ that only has the entries from $\mathcal{T}$, since all other entries are constrained to stay at 0. In order to do that, we will have to add additional constraints on β, to keep at 0 all variables β_j that correspond to features which were not selected by any weak classifier from $\mathcal{T}$.

The final optimization solution for iteration t of graph-regularized boosting procedure ProxGraphLogisticBoost is:

Step t.1: identify which direction in the optimization space is the $t+1$-th direction, by using the weak learner to train a weak classifier, which corresponds to selecting the weak classifier k with most negative value of

$$\frac{\partial \mathcal{R}}{\partial \phi_k}(\phi^t) = \frac{1}{m} \sum_{i=1}^{m} -y_i h_{ik} \mathcal{L}'(-y_i \sum_{j \in \mathcal{T}} \phi_j^t h_{ij}), \tag{6.54}$$

and treating that weak classifier as $t+1$-th classifier in the order of all weak classifiers. Add the classifier to set $\mathcal{T}$. Also, expand the set $\mathcal{T}$ to any additional classifiers if desired, and train them. In our experiments in this dissertation, we did not explore adding any additional classifiers, so we train only one weak classifier in Step t.1.

Step t.2: perform gradient step limited to all previously trained weak classifiers, including those just trained in Step t.1 by the weak learning algorithm

$$\psi^{t+1} = \phi^t - s^t \nabla_{\mathcal{T}} \mathcal{R}(\phi^t), \tag{6.55}$$

where

$$\nabla_{\mathcal{T}} \mathcal{R} = \left[\frac{\partial \mathcal{R}}{\partial \phi_j} \right], j \in \mathcal{T} \tag{6.56}$$

$$\frac{\partial \mathcal{R}}{\partial \phi_k}(\phi) = \frac{1}{m} \sum_{i=1}^{m} -y_i h_{ik} \mathcal{L}'(-y_i \sum_{j \in \mathcal{T}} \phi_j h_{ij}), \tag{6.57}$$

$$\mathcal{L}'(x) = \frac{e^x}{1 + e^x}, \tag{6.58}$$

Step t.3: perform a quadratic optimization step to obtain new vector ϕ^{t+1} representing the new classifier ensemble

$$\phi^{t+1} = \arg\min_{\phi} \quad \frac{1}{2s^t} ||\phi - \psi^{t+1}||^2 + c_1 \sum_{j=1}^{F} |\beta_j| + c_2 \sum_{j=1}^{F} \sum_{k=1}^{F} G_{jk} |\beta_j - \beta_k| \tag{6.59}$$

$$\text{subject to} \quad \beta = \Gamma \phi, \tag{6.60}$$

$$\beta_j = 0, \quad \text{if } \beta_j \text{ not used by any classifier from } \mathcal{T}, \tag{6.61}$$

where $\mathcal{T}$ is a set of all weak classifiers trained in previous boosting rounds and in Step t.1 of the current round t. c_1 and c_2 are user-defined constants that indicate the desired strength of regularization.

Within a single iteration t, Steps t.2 and t.3 may be repeated during the line search that selects the value of step size s^t that leads to lower value of the objective function.

6.3 Results and Discussion

We compared our graph regularized boosting method ProxGraphLogisticBoost results with classical AdaBoost as well as LogisticBoost. LogisticBoost is a boosting method that uses the same logistic loss function and line search as our proposed method, but does not involve the graph cut nor LASSO regularization achieved through proximal step. Because our Prox-GraphLogisticBoost is based on logistic loss function of logistic boost, LogisticBoost can serve

Table 6.1: Boosting results for Phospho200 Dataset

Method	AUROC	Feature count
ProxGraphLogisticBoost	0.824	40
LogisticBoost	0.766	33
AdaBoost	0.547	56

Table 6.2: Boosting results for PhosphoFull Dataset

Method	AUROC	Feature count
ProxGraphLogisticBoost	0.676	2
AdaBoost	0.632	69
LogisticBoost	0.535	2

as a baseline method to test the effect of graph and lasso regularization in Proximal Graph LogisticBoost.

For the Phospho200, we listed the area under ROC curve and the number of feature selected for each method in the table 6.1. Graph regularized LogisticBoost method has much higher AUROC value than LogisticBoost and AdaBoost. It also selects less features than AdaBoost.

PhosphFull dataset results are listed in table 6.2. Graph regularized LogisticBoost obtains much better AUROC value than AdaBoost and LogisticBoost. Also, both graph regularized LogisticBoost and LogisticBoost drastically reduce the feature size of AdaBoost method.

Phosph200 mutation dataset results are listed in table 6.3. Graph regularized Logistic-Boost obtains higher AUROC value than LogisticBoost, but not as high as AdaBoost. More-over, both Graph regularized LogisticBoost and LogisticBoost reduced AdaBoost's feature size to half.

Graph regularized ProxGraphLogisticBoost in general generates better results than Logis-ticBoost and AdaBoost. We also observe that Logistic loss function based Boosting methods tends to choose fewer number of features than classical AdaBoost, even though all three

Table 6.3: Boosting results for Phospho200 Mutation Dataset

Method	AUROC	Feature count
AdaBoost	0.646	65
ProxGraphLogisticBoost	0.577	32
LogisticBoost	0.516	31

methods were executed for the same number of rounds of boosting, 100, selecting one weak classifier in each round.

Chapter 7

Conclusions

The goal of our research is to invent algorithms to 1) integrate feature relations into machine learning methods; 2) build more efficient classifiers to separate samples of different classes; 3) generate sparse models that include fewer number of features; 4) discover the underlying differences between different classes.

To realize these goals, in this dissertation we proposed three conceptually distinct graph-based regularization machine learning methods: Graph Connectivity Constrained AdaBoost, Graph-Regularized Linear SVM and Graph-Regularized LogisticBoost. The first method is based on classical AdaBoost, and adds a hard constraint to it - the classifier has to represent a connected set of features. The other two methods represent a different approach. Instead of a hard constraint, they introduce a penalty that effectively promotes classifiers to be based on connected sets of features, but they do not strictly enforce that condition. One of the method is based on SVMs and the other on boosting, the two most prominent examples of state-of-the-art convex classification methods. Both of these penalty-based graph regularization methods involve convex extensions of submodular set functions. We derived them using one example of such a function, the graph cut, but they can work equally well with other submodular graph functions. By defining the third of our methods, the graph-regularized boosting, we have also provided a more general result. We have demonstrated how to extend boosting from coordinate-wise descent to more general proximal descent that can include non-smooth penalties.

We compared our methods with state-of-the-art methods on three simulated biological datasets: Phospho200, PhosphoFull, and Phospho200 Mutation. These dataset were constructed to cover scenarios with small and large number of features, and a simple and complex pattern of differences between classes. The new proposed methods almost always perform better than corresponding baseline methods that do not include graph-based regularization, obtaining higher predicting accuracy. These results show that the inclusion of graph-based regularization has met the goals we set out at the onset of the dissertation research.

Bibliography

[1] F. R. Bach. Structured sparsity-inducing norms through submodular functions. In *NIPS*, pages 118–126, 2010.

[2] M. Bailly-Bechet, C. Borgs, A. Braunstein, J. Chayes, A. Dagkessamanskaia, J.-M. François, and R. Zecchina. Finding undetected protein associations in cell signaling by belief propagation. *Proceedings of the National Academy of Sciences*, 108(2):882–887, 2011.

[3] A. Beck and M. Teboulle. Gradient-based algorithms with applications to signal recovery. *Convex Optimization in Signal Processing and Communications*, 2009.

[4] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[5] L. Breiman. Prediction games and arcing algorithms. *Neural computation*, 11(7):1493–1517, 1999.

[6] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[7] S. B. Cho, J. Kim, and J. H. Kim. Identifying set-wise differential co-expression in gene expression microarray data. *BMC bioinformatics*, 10(1):109, 2009.

[8] H.-Y. Chuang, E. Lee, Y.-T. Liu, D. Lee, and T. Ideker. Network-based classification of breast cancer metastasis. *Molecular systems biology*, 3(1), 2007.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[10] T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[11] S. E. Dreyfus and R. A. Wagner. The steiner problem in graphs. *Networks*, 1(3):195–207, 1971.

[12] N. Duffy and D. Helmbold. A geometric approach to leveraging weak learners. In *Computational Learning Theory*, pages 18–33. Springer, 1999.

[13] S. Dughmi. Submodular functions: Extensions, distributions, and algorithms. a survey. *arXiv preprint arXiv:0912.0322*, 2009.

[14] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, et al. Least angle regression. *The Annals of statistics*, 32(2):407–499, 2004.

[15] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[16] J. Friedman, T. Hastie, and R. Tibshirani. Special invited paper. additive logistic regression: A statistical view of boosting. *Annals of statistics*, pages 337–374, 2000.

[17] Z. Guo, Y. Li, X. Gong, C. Yao, W. Ma, D. Wang, Y. Li, J. Zhu, M. Zhang, D. Yang, et al. Edge-based scoring and searching method for identifying condition-responsive protein–protein interaction sub-network. *Bioinformatics*, 23(16):2121–2128, 2007.

[18] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *The Journal of Machine Learning Research*, 3:1157–1182, 2003.

[19] I. Guyon, J. Weston, S. Barnhill, and V. Vapnik. Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1-3):389–422, 2002.

[20] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[21] A. E. Hoerl and R. W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

[22] J. Hu, H.-S. Rho, R. H. Newman, W. Hwang, J. Neiswinger, H. Zhu, J. Zhang, and J. Qian. Global analysis of phosphorylation networks in humans. *Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics*, 1844(1):224–231, 2014.

[23] J. Hu, H.-S. Rho, R. H. Newman, J. Zhang, H. Zhu, and J. Qian. Phosphonetworks: a database for human phosphorylation networks. *Bioinformatics*, 30(1):141–142, 2014.

[24] T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(suppl 1):S233–S240, 2002.

[25] M. Kayano, I. Takigawa, M. Shiga, K. Tsuda, and H. Mamitsuka. Ros-det: robust detector of switching mechanisms in gene expression. *Nucleic acids research*, 39(11):e74–e74, 2011.

[26] V. Kecman and I. Hadzic. Support vectors selection by linear programming. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 5, pages 193–198. IEEE, 2000.

[27] P. Klein and R. Ravi. A nearly best-possible approximation algorithm for node-weighted steiner trees. *Journal of Algorithms*, 19(1):104–115, 1995.

[28] D. Kostka and R. Spang. Finding disease specific alterations in the co-expression of genes. *Bioinformatics*, 20(suppl 1):i194–i199, 2004.

[29] Y. Lai, B. Wu, L. Chen, and H. Zhao. A statistical method for identifying differential gene–gene co-expression patterns. *Bioinformatics*, 20(17):3146–3155, 2004.

[30] C. Li and H. Li. Network-constrained regularization and variable selection for analysis of genomic data. *Bioinformatics*, 24(9):1175–1182, 2008.

[31] K.-C. Li. Genome-wide coexpression dynamics: theory and application. *Proceedings of the National Academy of Sciences*, 99(26):16875–16880, 2002.

[32] M. Liu, A. Liberzon, S. W. Kong, W. R. Lai, P. J. Park, I. S. Kohane, and S. Kasif. Network-based analysis of affected biological processes in type 2 diabetes models. *PLoS genetics*, 3(6), 2007.

[33] D. Marbach, R. J. Prill, T. Schaffter, C. Mattiussi, D. Floreano, and G. Stolovitzky. Revealing strengths and weaknesses of methods for gene network inference. *Proceedings of the National Academy of Sciences*, 107(14):6286–6291, 2010.

[34] L. Mason, J. Baxter, P. L. Bartlett, and M. Frean. Functional gradient techniques for combining hypotheses. *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*, pages 221–246, 1999.

[35] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[36] R. E. Miller and J. W. Thatcher. *Complexity of computer computations.* Springer, 1972.

[37] M. Minsky and S. Papert. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, 19:88, 1969.

[38] R. H. Newman, J. Hu, H.-S. Rho, Z. Xie, C. Woodard, J. Neiswinger, C. Cooper, M. Shirley, H. M. Clark, S. Hu, et al. Construction of human activity-based phosphorylation networks. *Molecular systems biology*, 9(1), 2013.

[39] A. Y. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

[40] R. Nilsson, J. M. Peña, J. Björkegren, and J. Tegnér. Detecting multivariate differentially expressed genes. *BMC bioinformatics*, 8(1):150, 2007.

[41] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in optimization*, 1(3):123–231, 2013.

[42] E. Ravasz, A. L. Somera, D. A. Mongru, Z. N. Oltvai, and A.-L. Barabási. Hierarchical organization of modularity in metabolic networks. *Science*, 297(5586):1551–1555, 2002.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[44] I. Sadowski, B.-J. Breitkreutz, C. Stark, T.-C. Su, M. Dahabieh, S. Raithatha, W. Bernhard, R. Oughtred, K. Dolinski, K. Barreto, et al. The phosphogrid saccharomyces cerevisiae protein phosphorylation site database: version 2.0 update. *Database: the journal of biological databases and curation*, 2013, 2013.

[45] T. Schaffter, D. Marbach, and D. Floreano. Genenetweaver: in silico benchmark generation and performance profiling of network inference methods. *Bioinformatics*, 27(16):2263–2270, 2011.

[46] M. S. Scott, T. Perkins, S. Bunnell, F. Pepin, D. Y. Thomas, and M. Hallett. Identifying regulatory subnetworks for a set of genes. *Molecular & Cellular Proteomics*, 4(5):683–692, 2005.

[47] S. S. Shen-Orr, R. Milo, S. Mangan, and U. Alon. Network motifs in the transcriptional regulation network of escherichia coli. *Nature genetics*, 31(1):64–68, 2002.

[48] L. K. Southworth, A. B. Owen, and S. K. Kim. Aging mice show a decreasing correlation of gene expression within genetic modules. *PLoS genetics*, 5(12):e1000776, 2009.

[49] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. Biogrid: a general repository for interaction datasets. *Nucleic acids research*, 34(suppl 1):D535–D539, 2006.

[50] A. Subramanian, P. Tamayo, V. K. Mootha, S. Mukherjee, B. L. Ebert, M. A. Gillette, A. Paulovich, S. L. Pomeroy, T. R. Golub, E. S. Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences of the United States of America*, 102(43):15545–15550, 2005.

[51] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[52] A. Walley, P. Jacobson, M. Falchi, L. Bottolo, J. Andersson, E. Petretto, A. Bonnefond, E. Vaillant, C. Lecoeur, V. Vatin, et al. Differential coexpression analysis of obesity-associated networks in human subcutaneous adipose tissue. *International Journal of Obesity*, 36(1):137–147, 2012.

[53] L. Wang, J. Zhu, and H. Zou. The doubly regularized support vector machine. *Statistica Sinica*, 16(2):589, 2006.

[54] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

[55] T. T. Wu, K. Lange, et al. The mm alternative to em. *Statistical Science*, 25(4):492–505, 2010.

[56] Y. Xi, U. Hasson, P. J. Ramadge, and Z. J. Xiang. Boosting with spatial regularization. In *Advances in Neural Information Processing Systems*, pages 2107–2115, 2009.

[57] M. Xu, M.-C. J. Kao, J. Nunez-Iglesias, J. R. Nevins, M. West, and X. J. Zhou. An integrative approach to characterize disease-specific pathways and their coordination: a case study in cancer. *BMC genomics*, 9(Suppl 1):S12, 2008.

[58] E. Yeger-Lotem, L. Riva, L. J. Su, A. D. Gitler, A. G. Cashikar, O. D. King, P. K. Auluck, M. L. Geddie, J. S. Valastyan, D. R. Karger, et al. Bridging high-throughput genetic and transcriptional data reveals cellular responses to alpha-synuclein toxicity. *Nature genetics*, 41(3):316–323, 2009.

[59] A. Z. Zelikovsky. An 11/6-approximation algorithm for the network steiner problem. *Algorithmica*, 9(5):463–470, 1993.

[60] B. Zhang, S. Horvath, et al. A general framework for weighted gene co-expression network analysis. *Statistical applications in genetics and molecular biology*, 4(1):1128, 2005.

[61] J. Zhang, Y. Ji, and L. Zhang. Extracting three-way gene interactions from microarray data. *Bioinformatics*, 23(21):2903–2909, 2007.

[62] J. Zhu, S. Rosset, T. Hastie, and R. Tibshirani. 1-norm support vector machines. *Advances in neural information processing systems*, 16(1):49–56, 2004.

[63] H. Zou. An improved 1-norm svm for simultaneous classification and variable selection. In *AISTATS*, volume 2, pages 675–681. Citeseer, 2007.

[64] H. Zou and T. Hastie. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

919-600-1565
gaox2@vcu.edu
cissigao@gmail.com

Xi "Cissi" GAO

10 N. Calvert St Apt 705
Baltimore, MD 21202

SUMMARY

Data scientist currently working on improving machine learning classification methods by incorporating pre-existing graph knowledge of feature network structure. 5+ years of programming experience focused on bioinformatics high throughput data mining and network analysis.

WORK EXPERIENCE

Philips Healthcare
Data Scientist July 2011 – Present

EDUCATION

Virginia Commonwealth University (VCU), Richmond, VA
Ph.D. candidate in Computer Science Jan. 2011 – May. 2015
Proposal Title:
Graph-based Regularization in Machine Learning: Discovering Driver Modules in Biological Networks

Virginia Commonwealth University
Master of Science in Bioinformatics, Quantitative Statistical Track Aug. 2008 – Aug. 2010

Sun Yat-Sen University, Guangzhou, Guangdong, China
Bachelor of Science in Medicine, Preventive Medicine Track Sept. 2003 – Jun. 2008

RESEARCH EXPERIENCE

Computer Science Department, VCU (PI: Dr. T. Arodz) Jan. 2011 – May. 2015
- machine learning using ensembles and SVM incorporating graph knowledge

Center for the Study of Biology Complexity, VCU (PI: Dr. G. Buck) Jan. 2009 – Aug. 2010
- statistical analysis of DNA sequencing data

Virginia Institute for Psychiatric and Behavioral Genetics, VCU (PI: Dr. Z. Zhao) Jan. 2009 – May 2009
- development of alcoholism- and schizophrenia-related gene databases

TEACHING EXPERIENCE

Computer Science Department, VCU Aug. 2014 – May 2015
- teaching assistant: Introduction to Discrete Structures
- teaching assistant: Algorithm Analysis with Advanced Data Structures

School of Life Science, VCU Aug. 2008 – May 2009
- teaching assistant: recitation section instructor

AWARDS & HONORS

2013 Outstanding Early-career Student Researcher Award, Department of Computer Science, VCU
2014 Grace Hopper Celebration of Women in Computing Scholarship Grant

PUBLICATIONS

[1] X. Gao, E. Petricoin, K. Ward, S. Goldberg, T. Duane, D. Bonchev, T. Arodz, R. Diegelmann. ***A Proteomic analysis of systems biology of human wound healing.*** **Wound Repair and Regeneration** (under review).

[2] X. Gao, T. Arodz. ***Detecting Differentially Co-expressed Genes for Drug Target Analysis.*** In: "Proceeding of the 13th International Conference on Computational Science ICCS'2013", Procedia Computer Science 18:1392-1401, Elsevier, 2013.

[3] G. Federici, X. Gao, J. Slawek, T. Arodz, A. Shitaye, J. D. Wulfkuhle, R. de Maria, L. A. Liotta, E. F. Petricoin. ***Systems Level Network Analysis of the NCI-60 Cancer Cell Lines by Alignment of Protein Pathway Activation Modules with Multiple "-omic" Data Fields and Therapeutic Response Signatures.*** **Molecular Cancer Research**, 11:676-685, AACR, 2013.

[4] X. Gao, T. Arodz. ***Robust Differential Co-expression Discovery: an Insight into Pharmacodynamics of Tyrosine Kinase Inhibitor.*** In: Proceedings of the ACM Conference on Bioinformatics, Computational Biology and Biomedicine (ACM- BCB'12), pp. 604-606, 2012.

[5] X. Gao, T. Arodz. ***Unified Consensus Co-expression and Differential Co-expression Protein Networks.*** International Conference on Intelligent Biology and Medicine, ICIBM 2012, Nashville, TN (abstract & poster).

[6] W. Chen, X. Gao, C. Sun, W. Wan, D. Zhi, N. Liu, X. Chen and G. Gao. ***Evaluation of Association Tests for Rare Variants Using Simulated Data Sets in the Genetic Analysis Workshop 17 data.*** **BMC Proceedings,** 5(Suppl 9): S86, 2011.

[7] J. M. Fettweis, J. P. Alves, J. F. Borzelleca, J. P. Brooks, C. J. Friedline, Y. Gao, X. Gao, P. Girerd, M. D. Harwich, S. L. Hendricks, K. K. Jefferson, V. Lee, H. Mo, M. C. Neale, F. A. Puma, M. A. Reimers, M. C. Rivera, S. B. Roberts, M. G. Serrano, N. U. Sheth, J. L. Silbert, L. Voegtly, E. C. Prom-Wormley, B. Xie, T. P. York, C. N. Cornelissen, J. L. Strauss, L. J. Eaves, G. A. Buck. ***The Vaginal Microbiome: Disease, Genetics and the Environment.*** **Nature Precedings,** 2010.

INTERNSHIPS

Guangzhou Center of Disease Control, Guangzhou, Guangdong, China	Jan. 2008 – Jun. 2008
Epidemiology Lab, Sun Yat-Sen University	Sept. 2007 – Dec. 2007
North Canton Hospital, Shaoguan, Guangdong, China	Mar. 2007 – Aug. 2007
Sun Yat-Sen Hospital, Guangzhou, Guangdong, China	Aug. 2006 – Jan. 2007
Immunology Lab, Sun Yat-Sen University	Aug. 2005 – Jul. 2006

COMPLETED COURSEWORK (Selected)

Integrated Bioinformatics
Bioinformatics Algorithms
Network Biology
Microarray Analysis I & II
Database Theory
Knowledge Discovery & Data Mining
Advanced Algorithms
Statistical Learning and Soft Computing
Statistical Computing
Modern Regression and Statistical Modeling

PROGRAMMING AND COMPUTER SKILLS (Selected)

Java, Perl, R, MATLAB, Python, SAS, JMP, MySQL