THESIS

LEVERAGING EXPRESSION AND NETWORK DATA FOR PROTEIN FUNCTION PREDICTION

Submitted by

Kiley Graim

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Summer 2012

Master's Committee:

Advisor: Asa Ben-Hur

Chuck Anderson
Jeff Achter

ABSTRACT

LEVERAGING EXPRESSION AND NETWORK DATA FOR PROTEIN FUNCTION PREDICTION

Protein function prediction is one of the prominent problems in bioinformatics today. Protein annotation is slowly falling behind as more and more genomes are being sequenced. Experimental methods are expensive and time consuming, which leaves computational methods to fill the gap. While computational methods are still not accurate enough to be used without human supervision, this is the goal.

The Gene Ontology (GO) is a collection of terms that are the standard for protein function annotations. Because of the structure of GO, protein function prediction is a hierarchical multi-label classification problem. The classification method used in this thesis is GOstruct, which performs structured predictions that take into account all GO terms. GOstruct has been shown to work well, but there are still improvements to be made.

In this thesis, I work to improve predictions by building new kernels from the data that are used by GOstruct. To do this, I find key representations of the data that help define what kernels perform best on the variety of data types. I apply this methodology to function prediction in two model organisms, *Saccharomyces cerevisiae* and *Mus musculus*, and found better methods for interpreting the data.

ACKNOWLEDGMENTS

I want to send a heartfelt thanks to Mike Vogel and Artem Sokolov, who have both been so strong for me. Artem has been there for over four years as so many roles for me– a mentor, a lover, a confidant, a partner in research and at home. I could not imagine my life without him by my side. Mike has had an unusual role in my life; He has given so much of himself to others, and has helped me find a wonderful perspective on life.

Some of my closest friends have also been guiding forces in my life. Kate Ericson has been my best friend and a source of sanity for the past few years. I will never forget all the times we've knitted together at the department while we discuss our research, or gone to the gym to do the same. Thank you Dan Anderson for having helped me with my troubles, both in computer science and in other aspects of my life.

Inside of the department I have had so many people stand up for me. The other women in the department have shared my journey and have allowed me to be a part of theirs. As an undergraduate student, Michelle Strout and Dan Massey both gave me opportunities to do research and be involved in academia. While I was a graduate student, they stood up for me when they didn't have to, and I will never forget how supported and safe I felt because of them, during that time. Chuck Anderson and Wim Bohm are constant reminders that there are so many reasons to smile, and both taught me how to share my love for my work and for my field. My advisor, Asa Ben-Hur, has taught me to be a researcher, and for that I can't thank him enough. All of these people and others in the department have given me incredible opportunities.

Finally, I want to thank my family. My parents are the original reason I wanted to go to graduate school, and the reason that I love to teach. Thanks to them, I am a strong, independent woman.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

A protein is a sequence of amino acids. Protein function describes the role that a protein plays in the functioning of a cell. The roles that proteins play include carrying signals, contributing to cellular structure, binding to molecules, and binding to other proteins. Understanding protein function is key to understanding the biological processes occurring in an organism.

Protein function prediction is one of the prominent problems in bioinformatics today. Protein annotation is slowly falling behind as more and more genomes are being sequenced [11]. Experimental methods are expensive and time consuming, which leaves computational methods to fill the gap. While computational methods are still not accurate enough to be used in an unsupervised manner, this is the goal.

1.1 The Gene Ontology

The Gene Ontology (GO) provides a collection of terms that have become the standard in annotating protein function [32]. The creation of this ontology is a collaborative effort attempting to create naming standards, to ensure continuity in the field, since biological data is so diverse. GO is divided into three namespaces– biological processes (BP), cellular component (CC), and molecular function (MF). BP is a series of molecular functions or other events, CC describes the location in the cell, and MF illustrates the abilities that the protein has within the cell. BP and MF differ in that BP describes a series of events, whereas MF describes one of those events. The GO hierarchy forms a directed acyclic graph (DAG), and multiple terms can be used as annotations for a protein since a protein can perform multiple

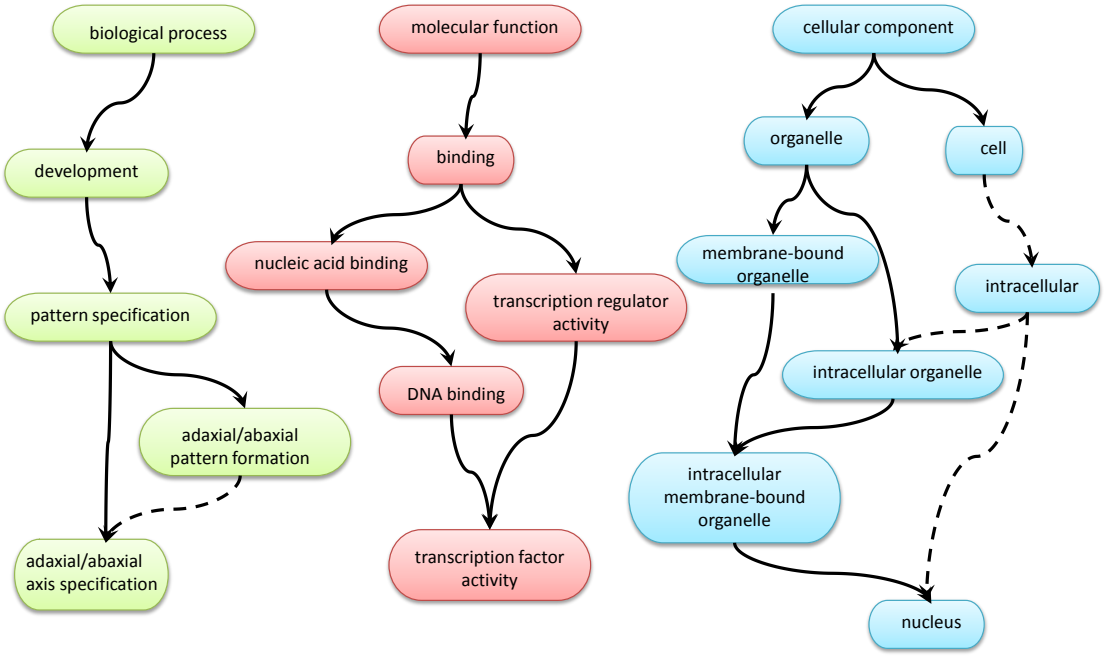


Figure 1.1: GO term hierarchy clusters and examples of terms within each cluster. (Reproduction of an image from Rogers *et al.* [40]).

functions. Figure 1.1 shows a small part of each of the three major groups of GO terms. Nodes lower in the GO hierarchy are more specific. Therefore, if any GO term is assigned to a protein, all of its parent nodes must by definition also be assigned in order for the prediction to be a valid description of the protein function. The DAG structure of GO gives meaning between the GO terms. I leverage this structure when making my predictions, by using other terms in the related subtree of a GO term to help classify that term. Figure 1.2 shows an example of the GO terms annotated to a sample protein. As shown in the figure, if a child node is set as an annotation for a given protein, that protein must also be annotated with the parent nodes.

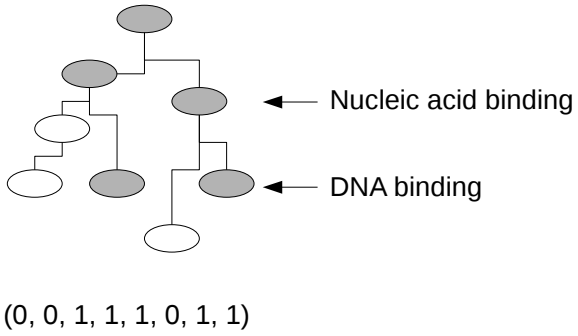


Figure 1.2: An example of the GO term hierarchy. A directed acyclic graph, all parent nodes must also be valid annotations if their children are. Taken with permission from Sokolov *et al* [47].

1.2 Formulation of the Problem

Protein function prediction is a hierarchical multi-label classification problem because of the structure of GO and because each protein may have multiple functions [2]. The traditional method for function prediction has been transfer of annotation from proteins with similar sequence or structure [31]. Transfer of annotation is successful with sequence data, but often not with network and other high throughput data because it is unable to handle the noise in such data. Many machine learning methods divide the problem into a series of binary classification issues [37, 38], one for each Gene Ontology (GO) term [32]. Some methods enforce congruency between the individual GO terms predictions, meaning that whenever a term is predicted, its parents must also be. The method used in this thesis is GOstruct [46], which takes a similar approach but instead of a series of binary classifications, it performs a structured prediction that takes into account all GO terms. GOstruct has performed strongly in both the CAFA Challenge (http://biofunctionprediction.org/) and MouseFUNC [39].

1.3 Data

A collection of different types of data are commonly used in protein function prediction. Protein sequence, Protein Protein Interactions (PPI), and microarray are some of the most common. Similarity in sequence is an indication of similarity in function and is a common approach for prediction of protein function [31]. PPI data tracks interactions between proteins [48, 21]. Microarray data, also called gene expression (GE) data, tracks the situations in which genes are expressed [42].

Using a naive approach to represent data in GOstruct is not effective for GE data. GE data is key due to the vast quantities of data available. Using the raw GE data only slightly improves GOstruct's performance. In this thesis, I use other methods to extract information from GE data. I experiment with biclustering and independent component analysis using GE data. GE data can benefit from biclustering, since biclustering can catch pockets of similarity in expression profiles. I use two different implementations of biclustering algorithms, and use the resultant bicluster data as features (instead of the gene expression levels). Doing this isolates more condition specific information and can reduce runtime resource requirements, by reducing the number of features.

Protein protein interaction data is the most predictive of all the data currently used by GOstruct. Network data is about paths, and in this thesis I explore ways of looking at the adjacency data and which path lengths are significant. While it is the best data for predictions that I currently use, my goal is to improve this by modifying both the input data, and the ways we look at that data. For the first method, I change the PPI data matrix in an operation similar to the graph laplacian. Doing this emphasizes direct links between individual proteins, increasing the perceived similarity between the protein pairs. For the second method, I apply the diffusion kernel to PPI data [25, 49], a method that combines interaction network paths of all lengths.

In this thesis, I work to improve predictions by manipulating the data representations that are used by the structured SVMs, given the same input data. To do this, I more fully study the data sets and find key components of them which help define what methods would

perform best on the variety of data types. This is done over several species and data types, ensuring that the results are not data set specific.

My thesis work shows that, of the GE kernel methods, independent component analysis performs the best. It outperforms the current method used to build GE data kernels in GOstruct. Several of the other methods used show promise when combined. For the network data, I find that none of the new methods show significant improvement. I opt to continue using the network data as we have been, but show how these other methods can be utilized in other prediction problems. Although the network based kernel methods used in this thesis do not improve GOstruct performance, in the following chapters I will explain the benefits of each and show how they perform. These methods can be useful in other scenarios, and it is important to understand why they are not helpful here.

Before explaining this I walk through how GOstruct works. I do this in Chapter 2, where I first tackle binary and structured support vector machines. With a firm understanding of these we can then continue to explore the specifics of GOstruct and how it works. In Chapter 2 I also explain the forms of the data that I use in this thesis– their origin, meaning, form, and importance. Once we have gone over the background of the problem, I continue on to discuss the methods used in this thesis. Chapter 3 walks over each method in detail and shows how they are applied to the data. Here I explain how experiments are set up, what the results from them mean, and how they are found. I show and interpret the results from those experiments in Chapter 4. At this point I will talk about specific GO term prediction results as well as overall results. With this information, I present my conclusions and future plans in Chapter 5.

Chapter 2

Background

GOstruct is a structured output approach to protein function prediction. It determines collections of GO terms that should be assigned to each protein in an input set, and takes into account the structure of GO while doing so. In order to introduce GOstruct I first explain support vector machines, then structured support vector machines.

2.1 Support Vector Machines

The support vector machine (SVM) is a supervised learning method [5], commonly used in bioinformatics problems partially because of their ability to handle high dimensional data [22, 3]. Given an input set of features and samples, an SVM will build a model for distinguishing between two classes. As shown in Figure 2.1, binary SVMs calculate a hyperplane that maximizes the margin between the two classes. For a linear, binary SVM, we want to learn a discriminant function:

$$f(x) = w^T x + b, \tag{2.1}$$

where w is a weight vector and b is a bias term. The sign of $f(x)$ is what determines which class x belongs to. For large margin problems, we have a training set $(x_i, y_i)_{i=1}^n$, with a vector of terms x_i and labels y_i with ± 1 values. The large margin machine learning problem can be expressed as:

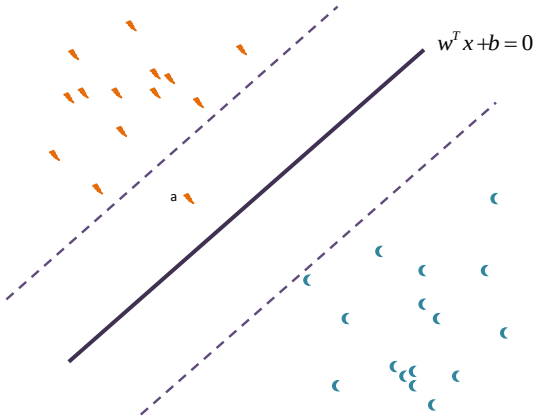


Figure 2.1: The hyperplane created by an SVM, separating two classes. The point a is an example of a margin violation. Soft margin SVMs allows violations like this so that the margin can be significantly larger. There is a user-specified parameter that handles the balance between the number of margin violations and the size of the margin.

$$\min_{w,b}(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i)$$
$$\text{subject to: } y_i(w^T x_i + b) \geq 1 - \xi_i, i = 1, ..., n$$
$$\xi_i \geq 0, i = 1, ..., n. \tag{2.2}$$

The first term ($\frac{1}{2}\|w\|^2$) maximizes the margin, and the second term ($C\sum_{i=1}^{n}\xi_i$) minimizes the margin violations, making the classifier fit the data.

To obtain a nonlinear classifier we use a mapping of ϕ, to a possibly higher dimensional feature space where classification is performed:

$$f(x) = w^T\phi(x) + b. \tag{2.3}$$

To not have to calculate this feature mapping, we use kernels. Kernels define similarity between samples through dot products by taking the dot product of the projection of those samples into the feature space:

$$K(x, x') = \phi(x)^T\phi(x') \tag{2.4}$$

By doing this we can compute the value of the dot products in our feature space without having to explicitly map into that possibly high dimensional space. In many cases we can use kernels so that we do not have to explicitly calculate the values for Equation 2.4 [43].

2.2 Structured Support Vector Machines

Figure 2.2 demonstrates several samples being classified, and the margin separating out the true values. The goal of structured support vector machines (SSVMs) is to map some input space $\mathcal{X}$ to some output space $\mathcal{Y}$ based on a training set of input-output pairs. SSVMs learn a discriminant function $F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ that maps input-output pairs over some real space, maximizing the margin based on a training set. F is assumed to be a linear function in a feature space defined in the joint input-output space:

$$F(x, y) = w^T \psi(x, y). \tag{2.5}$$

SSVMs follow the same theory as SVMs, however they classify samples that belong to arbitrary discrete spaces. SSVMs use a fitness function to differentiate between the different classes; they build a model that maximizes the size of the margin while still complying with that fitness function.

The feature map ψ is user defined, and the weight vector w is trained by the SSVM. The objective is to maximize the compatibility scores for all training examples with the correct label. Similar to SVMs, SSVMs define a margin, which should minimize margin violations, to separate true labels from false for all examples:

$$\arg\max_{y \epsilon \mathcal{Y}} w^T \psi(x_i, y) = y_i \text{ for } i = 1, ..., n \tag{2.6}$$

It isn't always possible to do this, so we instead solve:

$$\min_{w,\xi}(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i)$$

$$\text{such that } w^T\psi(x_i, y_i) - \max_{y \epsilon \mathcal{Y} \setminus y_i} w^T\psi(x_i, y) \geq 1 - \xi_i \text{ for } i = 1, ..., n \tag{2.7}$$

Equation 2.8 has $\Delta(\mathbf{y}_i, y)$ instead of 1 like Equation 2.7 because we don't need the same size margin for all of the output space points.

$$\text{such that } w^T\psi(x_i, y_i) - \max_{y \epsilon \mathcal{Y} \setminus y_i} w^T\psi(x_i, y) \geq \Delta(\mathbf{y}_i, y) - \xi_i \text{ for } i = 1, ..., n \tag{2.8}$$

The goal is to maximize the size of the margin between points in the output space $\mathcal{Y}$ with the fewest possible margin violations. Where SVMs use kernels with the input space data only (Equation 2.4), SSVMs must also consider the output space data because ψ is a function of both the input and output. With this in mind, we reformulate the SSVM equivalent of Equation (2.4) to be:

$$K((x, y), (x', y')) = \psi(x, y)^T\psi(x', y') \tag{2.9}$$

2.3 GOstruct

GOstruct is the SSVM I use in this thesis. There are three major components to GOstruct– an argmax computation, a kernel, and loss. The purpose of the discriminant function and kernel was explained in Section 2.2.

GOstruct uses a joint linear kernel which is a product of two linear kernels– one for the input space and another for the output space:

$$K((x, y), (x', y')) = K_x(x, x')K_y(y, y'). \tag{2.10}$$

For K_y we use a linear kernel. Kernels are used to map feature spaces, and the linear kernel is one of the most common. It works by computing the dot product of two samples and returning the value as the similarly measure. In this thesis I introduce 6 kernels, which are described in Sections 3.1 and 3.2.

GOstruct requires a loss function that is used for training, and I use one that is appropriate for hierarchical classification problems:

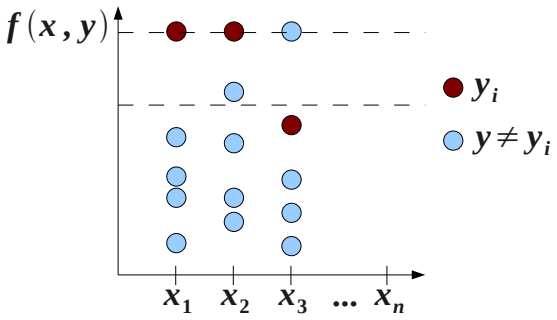


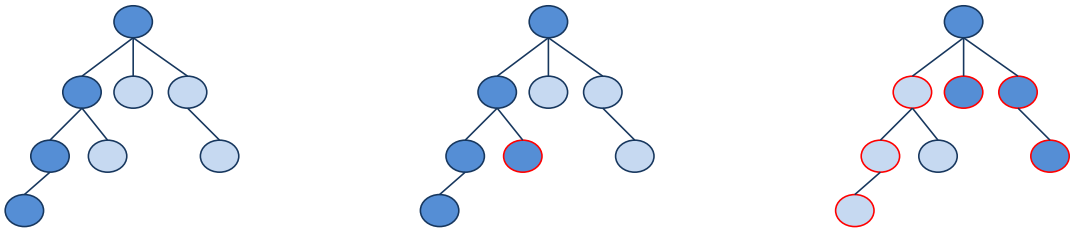
Figure 2.2: Taken with permission from Sokolov *et al* [47], this depicts constraints for the mapped feature space. Training samples are along the y-axis and label fitness on the x-axis. Label fitness means how correct a structured prediction is. The goal here is to have the highest compatibility value between true labels and examples with the largest possible margin separating those true labels from the other labels. This separation is shown by the two dotted lines. x_1 satisfies this, while example x_2 has a margin violation, and x_3 is misclassified.

$$\Delta(y, \hat{y}) = 1 - \frac{2y^T\hat{y}}{y^Ty + \hat{y}^T\hat{y}}, \tag{2.11}$$

which was proposed by Tsochantaridis *et al.* [51], and is related to the F_1 measure of accuracy used in information retrieval.

Classifier performance is often measured based on how many inaccurate predictions were made. This is straightforward in a binary environment where each prediction is either completely correct or completely false, however there are more factors to consider in the GO namespace. A prediction is a collection of GO terms in the GO space, and having even one incorrect prediction (whether predicting that a term should be associated with a given protein or not) should be marked as incorrect. This leads to a variety of degrees of incorrectness in predictions, as is shown in Figure 2.3. In this figure one sees how it can be much more desirable to incorrectly annotate one specific term in the structured prediction, and get the rest of the predictions correct, than to incorrectly designate many of those terms. Our goal is to minimize the level of incorrectness in our predictions. The loss function is Equation 2.11 is appropriate because it achieves this, whereas binary loss does not.

In this thesis I work predominately with two major types of data–GE and network(protein-protein interaction(PPI)) data. For the network data there are two new methods, one is a



(a) Correct (b) Slightly Wrong (c) Very Wrong

Figure 2.3: Three examples of error in predictions. 0-1 loss does not differentiate between the slight misclassification in 2.3(b) and 2.3(c), although 2.3(b) is only wrong in a few cases whereas 2.3(c) is completely wrong. Subfigure 2.3(a) shows a completely correct prediction.

new kernel type and the other is a data preparation method. Each of these is intended to emphasize different paths in the network data. I discuss these methods in Section 3.1. For the GE data there are four new methods that are all data preparation changes. I explain each of these methods and their uses in Section 3.2.

2.4 Data

In this thesis I work extensively with different types of data from multiple origins and species. *Mus musculus* (mouse) and *Saccharomyces cerevisiae* (yeast) are the organisms used for experiments performed. These were chosen for several reasons. I chose *S. cerevisiae* because it is a model organism, because of the simplicity of its structure. *S. cerevisiae* is a commonly chosen organism for bioinformatics problems because it has been studied extensively by biologists. *M. musculus* is also a model organism, and was chosen because of its importance in research. There is extensive work being done on both organisms.

There are several standard and commonly available data types used here. PPI data is a collection of proteins and all proteins that they are known to interact with. For each protein, there are proteins with which it interacts. GE data is a measure of the levels of mRNA in a specific set of conditions. Finally, sequence based data is a collection of protein sequences (the strings of amino acids that a protein is composed of). All of these data types have different strengths and weaknesses, which are described in more depth in the following sections.

2.4.1 Protein-Protein Interaction

Kernels built on PPI data try to capture the observation that if two proteins interact, they often share a similar function [54, 31]. Such data is generally stored in graph form, where nodes are proteins and edges are the interactions between proteins (see Figure 2.4). Today there are high-throughput interaction data available from several databases. The interaction data for this thesis is taken from the General Repository for Interaction Data sets (BioGRID) [48] and the Search Tool for the Retrieval of Interacting Genes/Proteins (STRING) [21].

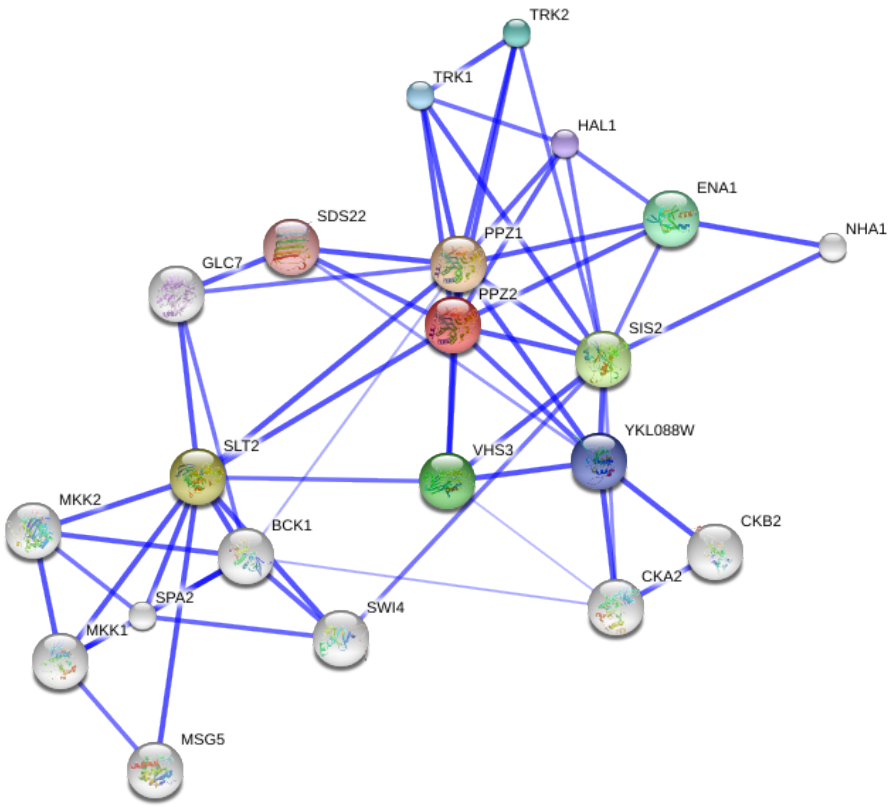


Figure 2.4: Interaction neighborhood for the protein S000002844 (*Serine/threonine-protein phosphatase*) in *S. cerevisiae*. Edge width indicates the confidence level, thicker edges meaning higher confidence. This graph was generated by STRING [21].

Not all of the organisms used in this work are characterized equally well. As such, there is a variety of information available for each. *S. cerevisiae* has the most interaction information, with approximately 6,000 proteins listed and 163,000 interactions [48]. From the same source,

M. musculus has much less information: 2,300 interactions among 3,000 out of the 20,000 or so proteins.

The edges between protein nodes are weighted by the confidence that those two proteins interact. Confidence scores are determined by comparisons to a trusted association set [53], and an edge weight of zero signifies that there is no known interaction between the two proteins.

2.4.2 Gene Expression

Aside from sequence and interaction similarities, one can ascertain protein involvement in biological processes by checking the expression levels of genes in specific circumstances [54]. Gene expression data is most commonly represented with a heat map. In these maps, rows correspond to genes and columns to experiments (see Figure 2.5). This underlying data is often referred to as microarray data because the expression levels are measured on DNA microarray chips [42]. DNA microarray chips are layered with a set of complementary DNA (cDNA) strands, designed to bind to specific genes, and attached to the microarray chip through robotic printing. A fluorescing agent on the chip and fluorescent labels allow microarray data to measure expression levels for many genes in parallel. An example of microarray data is shown in Figure 2.5. Gene expression data tends to be very noisy, so methods that use this data must be very robust with respect to noise.

Data used in this thesis for *S. cerevisiae* and *M. musculus* originate from the Platform for Interactive Learning by Genomics Results Mining (PILGRIM) based at Princeton [13]. *S. cerevisiae* data contains information on 2,500 genes in 80 different experiments while *M. musculus* data has information on approximately 30,000 experiments.

2.4.3 Sequence

Proteins are sequences of amino acids folded into three dimensional structures. The sequence and structure of the protein define its function. Thus, one can take sequence data and perform transfer of annotation [31], where keywords are transferred from a known protein to proteins with a similar sequence. This is often done with sequence alignment tools, such as BLAST.

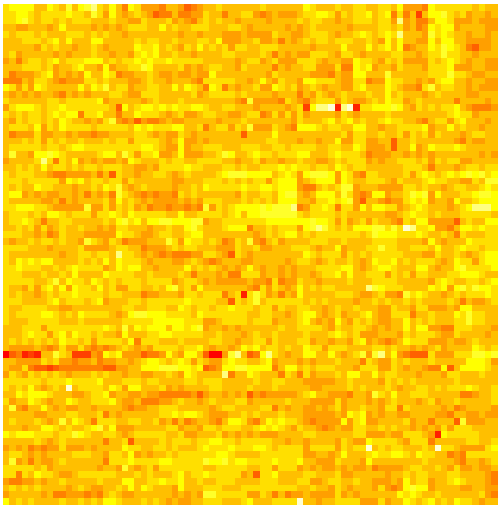


Figure 2.5: A heat map for a subset of the *S. cerevisiae* GE data.

Transfer of annotation is based on the assumption that proteins with similar sequence share functionality. Many studies have shown that this assumption is useful, but has limitations [4, 6, 10, 41], particularly in instances where single point mutations and gene duplication can lead to different protein functions despite similar sequence. Proteins often have several different functions only some of which are conserved across homologs (proteins sharing a common ancestor). Incorrect annotations exist in databases [6], and these annotations can be incorrectly transferred to other proteins. Despite these issues, transfer of annotation is still commonly used [14, 15, 34, 55].

Chapter 3

Methods

The data described in Chapter 2 is used to make kernels for the $K_x(x, x')$ part of Equation 2.10. Usually this is a combination of kernels:

$$K_x(x, x') = K_{GE}(x, x') + K_{PPI}(x, x') + K_{Seq}(x, x') \tag{3.1}$$

Adding kernels together is essentially appending the feature spaces together [44]. In this thesis I run experiments with both combinations of kernels and individual kernels. This chapter describes the kernel methods used in this thesis in detail. I explain how each of these methods works and show how it applies to protein function prediction. The chapter is divided into four major sections– one for each of the data groups (network, expression, and sequence) and a final section for the experiment setup and an explanation of how I evaluate the experiment results.

3.1 Kernels for Network Data

3.1.1 Normalizing Data

I remove terms that are not well represented, as was done on our previous work [45]: Any terms with fewer than 10 mentions are removed from the annotations file and data sets. This is done as part of the preprocessing, before kernels are built and tests are run. The samples eliminated by this preprocessing step are completely removed from consideration.

3.1.2 Linear Kernels Based on the Adjacency Matrix

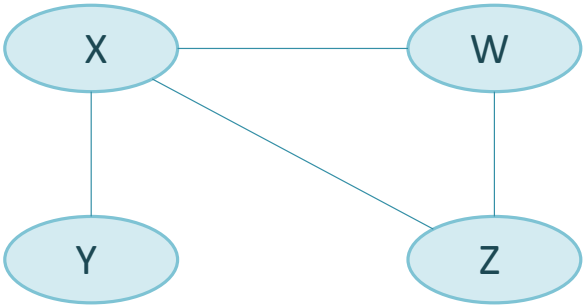
PPI data performs strongly for protein function prediction [8, 31, 35, 52, 54]. By applying some simple manipulations to the data, in theory I can improve PPI's contribution to predictions. PPI data comes in the form of an adjacency matrix. Generally PPI data records the confidence that there is interaction between proteins, although sometimes it is simply a binary record (interacts/does not interact). Since most proteins do not interact with themselves, the diagonal of this adjacency matrix is primarily zeroes. In my adaptation (called PPI-DIAG) I change the diagonals of the matrix to imply the highest level of interaction.

Although the linear kernel does not innately find such connections, they are instructive when using PPI data to assign annotations. The example below highlights the importance of finding immediate neighbors. Out of the adjacency matrix (A) I construct the following kernel:

$$K = A^T A$$

Each entry K_{ij} is the amount of overlap between the proteins i and j (meaning the amount of overlap in their interactions with other proteins). A and A^T are the same, so $A^T A$ is the same as squaring the matrix, where the result would consist of all paths of length two in the adjacency matrix. By changing the values of the diagonals, I am essentially overlaying a network of length one paths on top of the linear kernel. The potential problem with squaring the matrix is in the case where the proteins interact with each other but do not have any other proteins in common with which they react. In these cases, the linear kernel $A^T A$ would show no similarity between the two proteins even though they directly interact with each other. By changing the diagonals of the matrix (changing the adjacency matrix to show every protein to have a high confidence that it interacts with itself), I can fix this issue. Paths of length one would appear to have a length of two, since there is now a path from each node to itself. It also emphasizes direct interaction links, since the path of length one from protein X to protein Y would now appear to be two paths–one originating from X and

the other from Y. These paths would seem distinct since they must traverse the origin node first, thus changing the first hop in the path. In essence, the linear kernel is now finding all paths of length one in the adjacency matrix as well as the paths of length two that it normally finds in PPI data. Figure 3.1 illustrates an instance where this phenomenon would be helpful in predictions.



	X	W	Y	Z
X	-	1	1	1
W	1	-	0	1
Y	1	0	-	0
Z	1	1	0	-

Figure 3.1: An example of PPI interactions. In this example, the nodes are representative of proteins and the edges indicate interaction between proteins. This highlights the importance of catching both immediate neighbors and more distant neighbors.

The '-' in the matrix of Figure 3.1 are traditionally filled in with a zero, since none of the nodes has a path to itself. If one applies a linear kernel to this example, it gives all nodes that are connected by a path of length two. For example the value of the kernel with respect to X and Y is:

$$K(X,Y) = [0,1,1,1]^T[1,0,0,0] = 0$$

Although there is an edge between X and Y, the linear kernel finds no similarity between the two. One can easily argue that there is indeed some sort of relationship between the two nodes because of the edge between nodes X and Y. To adjust for this I modify the original adjacency matrix and replace each '-' with a one instead of a zero. This changes our example, giving a preferable solution where X and Y are considered related:

$$K(X,Y) = [1,1,1,1]^T[1,0,1,0] = 2$$

By making this change, we are finding how many paths of length one or two there are from X to Y. The shared edge is counted in both directions, resulting in the value of two even though there is only one distinct path. Since this manipulation essentially adds an edge from each node to itself, the linear kernel is finding two distinct paths- $X \rightarrow X \rightarrow Y$, and $Y \rightarrow Y \rightarrow X$.

3.1.3 Diffusion Kernel

The diffusion kernel works by considering paths of all lengths [25]. Past works have shown it to be effective on PPI data for protein function prediction problems [27, 46, 49].

$$K = e^{\beta H} \tag{3.2}$$

β is a user defined parameter. The diffusion kernel makes use of graph laplacian. In graph laplacian the diagonals of an adjacency matrix are changed to be the negative sum of the number of paths in each row of the matrix. Graph laplacian is performed on the PPI adjacency matrix. This changes the adjacency matrix, making the diagonals represent how connected each node is in the graph (H in Equation 3.2). The adjusted matrix is used by the diffusion kernel–it multiplies this matrix by the parameter β and take the exponential of that result. This result is the diffusion kernel matrix, which is then read in by GOstruct.

As described in Section 3.1.2, the linear kernel performs by finding all paths of length two between pairs of nodes and using the number of paths to determine how likely it is that the node pairs are similar. PPI-DIAG finds all paths of length one or two. The diffusion kernel builds on this–instead of finding all paths of length one or two, the diffusion kernel finds all paths of any length. Since it is computationally infeasible to find all possible paths of any length in an adjacency matrix, the diffusion kernel is actually findig an approximation. Paths may be infinitely long, and we will be able to find infinite paths as long as there is some connectivity in the graph.

To get a better understanding of what the diffusion kernel is doing, the matrix exponential can be approximated by using a Taylor series expansion:

$$e^A = \sum_{k=0}^{\infty} \frac{1}{k!} A^k = I + A + \frac{A^2}{2!} + \frac{A^3}{3!} + ... \tag{3.3}$$

Remember that $(A^k)_{ij}$ is the number of paths between proteins i and j of length k.

There are many ways to compute the exponential of a matrix [36]. I chose to use the *expm* package in R, which approximates the exponential of a matrix by using Ward's diagonal Pade approximation [12, 36]. I found this implementation to be beneficial on PPI data. I perform initial tests on the *S. cerevisiae* PPI data to establish the ideal value of β(I use the β from the experiment that has the best results), then use this same value for all experiments that include the diffusion kernel. The linear and diffusion kernels inspired my work with changing the PPI data matrix diagonals to show a high level of interaction between every protein and itself.

3.2 Kernels for Expression Data

Linear kernels built from raw GE data were not very helpful in preliminary experiments. In this thesis I use two different forms of biclustering, and independent component analysis on GE data, in the expectation that the different methods will reduce the noise in the GE data and highlight the key information instead. These methods are known for their ability to reduce the impact of noise in data, and I will explain why this is when I described them

Table 3.1: Information about the GE data.

	# Samples	# Features	# GO Terms
Datasets			
S. cerevisiae	2463	79	4852
M. musculus	5692	29144	7753

Table 3.2: The number of biclusters found on the *S. cerevisiae* GE data by each of the biclustering methods.

	# Biclusters
ISA	1320
Q1	544
Q2	999

in more detail.

Table 3.2 shows basic statistics for the *S. cerevisiae* GE data. The number of biclusters found by each of the biclustering methods is shown in Table 3.2. ISA found several hundred more biclusters than Q2, which find nearly twice as many biclusters as Q1.

3.2.1 Biclustering

The kernels used here work on GE data by comparing similarity of the entire expression profile for proteins in the microarray. Since proteins are often co-expressed in very specific situations, the linear kernel can easily miss a certain portion of a pair of expression profiles that are highly similar because the other areas of those two proteins may have very different expression levels. If there is a large area of the profile that is dissimilar, it can overwhelm the kernel result to show very little similarity between the proteins. Biclustering avoids this issue by dividing the expression data into groups of similarity by both row and column (across experiments and genes). By performing biclustering on a data set and using the bicluster information to build a linear kernel, one can capture condition specific gene co-expression instances. Biclustering has been shown effective for protein function prediction problems in the past [19, 20, 29].

Biclustering works similarly to clustering, however it clusters on both rows and columns of the data. Since protein interactions are often very specific to both the proteins involved and the scenarios they are in, biclustering can more accurately capture the protein interactions. I use two different biclustering methods– the *Iterative Signature Algorithm* (ISA) [19, 20] and the *QUalitative BIClustering algorithm* (QUBIC) [29].

Run time for linear kernels built from GE data becomes unreasonable as the data grows in size. By performing biclustering on GE data and using the bicluster data in the kernels, I avoid run time issues while still retaining pertinent information. This is true because I now use biclusters as features, instead of expression levels. Data is biclustered, then stored in another matrix where the proteins are samples and the features are the biclusters. This reduces the dimensionality of the problem, since we now have a collection of bicluster memberships as the features instead of every experiment. At most we can have as many biclusters as experiments (although it is highly unlikely to be that many). Genes track bicluster participation with ones and zeroes, a one meaning it is part of that bicluster and a zero otherwise. I construct data files for both of the biclustering algorithms that I use in this thesis, and use these to build linear kernels.

ISA and QUBIC were chosen due to the minimal overlap in their strengths. ISA first clusters on rows then on columns, whereas QUBIC first detects similarities in columns, then in rows, and again in columns. Both algorithms averaged fewer biclusters than other algorithms (for example FABIA [16] and SAMBA [50]), however QUBIC generally finds more than ISA. This is partially because QUBIC can detect and cluster on inverse co-expression.

I chose algorithms that produce a smaller number of biclusters, to enforce a high level of significance in bicluster participation. Neither algorithm produces so few biclusters as to cause concern that important information is being missed. QUBIC is deterministic while ISA is not, differentiating the two even more. Both algorithms are efficient, ensuring speedy execution. They are also both easy to use and simple to install, with exemplary instructions. This makes it harder to incorrectly use the algorithm, giving more credence to the results (since they are more likely to be legitimate results from the algorithm and not generated by

some issue with the setup). Finally, both have been used in other bioinformatics work and have been shown to be effective forms of biclustering for GE data [29, 20].

3.2.1.1 Iterative Signature Algorithm

The *Iterative Signature Algorithm* (ISA) is a biclustering algorithm developed specifically for GE data sets. It is built on the Signature Algorithm [19] by adding an iterative component to the initial step. Understanding the Signature algorithm makes ISA easier to understand.

ISA requires that input data be in matrix form. For GE data each column is a condition and each row is a specific gene, which can be represented easily in matrix form where each gene/condition is assigned to a certain row/column, respectively. The Signature Algorithm uses this data matrix to generate two normalized matrices, each with zero mean and variance, one with respect to gene and the other with respect to condition. Using these matrices, the Signature Algorithm then computes scores for each condition in the data set, using the average expression change over the set of proteins for that condition. At this point the most statistically significant conditions are selected and stored in a group (S_C), which is used when calculating the gene scores in the next step.

In the second major step of the Signature Algorithm, scores are computed for each gene in the data set using the previously computed conditions scores and a similar process to compute gene scores. The collection of gene scores is combined into one group that is used to construct signatures for each entry in the input matrix, used later in the algorithm for similarity comparisons between entries. Finally clusters are formed based on the signatures that have been found. To do this, the Signature Algorithm takes the gene/condition pair that has the highest signature intercept. This pair becomes the origin of a new module, and all pairs within a given threshold are added to that module. Distance from a given module takes into account both the distance from the origin and the distance from all other entries in a module, so that if a module ends up migrating slightly, entries will still be accurately placed. This step is repeated until all entries in the data set (all gene/condition pairs) are added to a module. Each entry will be in exactly one module.

The Iterative Signature Algorithm builds on the Signature Algorithm. It takes a random pair of entries and uses them as the origin of the first cluster. From here the algorithm is performed as normal, adding the closest entries (as defined by the signatures that are computed in the Signature Algorithm) until none are within the given threshold. Another set of random entries is chosen and the process is repeated until all terms have been assigned to a bicluster.

Algorithm 1 Iterative Signature Algorithm

```
Choose random gene set, G^(0)
while G^(n+1) ≠ G^(n) do
  Generate E_G^gc, E_C^gc
  Calculate S_C
  Find experiment signature
  Calculate S_G
  while unassigned values do
    Choose pair with highest signature intercept
    for i in entries do
      if i ≥ threshold then
        Add i to bicluster
      end if
    end for
  end while
end while
```

Both the Signature Algorithm and ISA allow cluster fusion. This means that clusters that are very close together can be combined into one larger cluster. By doing this, ISA and its predecessor enable the user to have more control over the algorithm. While certain cases require fewer biclusters, in protein function prediction I prefer to have the extra discrimination between proteins and conditions. After all entries in the matrix have been assigned to a bicluster, the Signature Algorithm then fuses together biclusters that have a high amount of overlap. Clusters can be generated that occupy a similar space and if the user wants fewer clusters then these can be merged together. For the experiments in this thesis, I do not perform this step.

3.2.1.2 Qualitative Biclustering Algorithm

The *Qualitative Biclustering algorithm* (QUBIC) [29] is the other biclustering algorithm I used in this thesis. QUBIC depends on a calculated *consistency level* to determine whether or not certain proteins and conditions should be included in a bicluster. Consistency level is defined as "the minimum ratio between the number of identical non-zero integers in a column and the total number of rows in the sub-matrix" [29]. QUBIC reads in the GE data and generates a weighted graph where nodes are genes and the edges are the similarity level between nodes:

$$e(g_1, g_2) = g_1^T g_2$$

Once this graph has been composed the real bulk of the algorithm begins. First the edges are put into a set S of possible seeds sorted by weight, where a larger weight is better. These seeds will be used to create new biclusters. The seed list is iterated over until empty. A given edge can be a bicluster seed if either one of the proteins is not in a bicluster already or the two proteins are not in the same bicluster.

Once a seed has been chosen the list of proteins is iterated over and all proteins who do not bring down the consistency level are added to the bicluster. Next the columns of the current bicluster are evaluated, and all columns are retained which meet the consistency level. Finally, QUBIC adds oppositely regulated proteins by comparing the proteins in the same manner as before but inversing the sign on one gene to check for equal but opposite expression levels.

I build two types of kernel from the QUBIC analysis, because QUBIC stores which proteins are oppositely regulated in the cluster, and which are not. Both kernels retain all of the bicluster data. The first kernel (Q1) uses the full bicluster as a feature, including the regularly and oppositely expressed proteins. The second kernel (Q2) divides the biclusters into two features, splitting these groups.

Algorithm 2 QUalitative BIClustering Algorithm

```
Build sorted seed list S_{g_i g_j}
while size(S) ≥ 0 do
  Choose seed, remove seeds until find seed s s.t.
Require: g_i or g_j is not in an existing bicluster OR
Require: g_i and g_j are in different biclusters
end while
while g_i ≥ consistency level do
  Add g_i to current bicluster
end while
for all columns do
  if column consistency ≥ bicluster consistency then
    Add column to bicluster
  end if
end for
for all proteins do
  Compute negative correlations and add to bicluster
end for
```

3.2.1.3 Using ISA and QUBIC

Neither ISA nor QUBIC produce results in a form that can be utilized by GOstruct. ISA is also a non-deterministic algorithm (owing to the randomly chosen bicluster seeds), and to ensure reliable results I ran ISA five times on each data set then combined the results from all five runs into one kernel. QUBIC is deterministic and only needed to be run once.

For each gene I mark which biclusters it is in by adding it to a list of biclusters following the name. Information is stored so that rows are gene names and columns are bicluster names. For ISA I create unique bicluster names and for QUBIC I keep the existing bicluster names.

Once these files were generated, I construct a linear kernel based on the bicluster data. With the data is stored in bicluster/gene format, the linear kernel can capture condition specific co-expression and report more accurate predictions, allowing me to maintain GE information while capturing the specific co-expression similarities.

3.2.2 Independent Component Analysis

Independent Component Analysis (ICA) is an unsupervised method known for its use in signal separation, EEG, and ECG data. ICA takes data and separates it into independent components, meaning it takes the features of a data space and projects them into a space that leaves as little correlation between separate components as possible. This linear transformation separates out the components. By creating a projection of independent components, ICA spreads data to find important hidden factors in it.

Take a data matrix X, where each row is a specific gene and columns are experiments applied to those proteins. ICA takes this and computes a matrix to linearly transform the matrix X to isolate the source components. This transformation takes the form:

$$X = AS, \tag{3.4}$$

where the matrix S contains the independent components built from X, and A acts as a weight matrix.

In this thesis I use fastICA [33] to perform ICA on GE data. FastICA assumes that the input matrix X is preprocessed, meaning that it expects data to be normalized. It returns the mixing matrix A, and as well as the matrix S of sources (which can be used for feature selection). Once ICA has been performed, the mixing matrix can be used in predictions.

ICA is canonically used in signal separation tasks. An example is of conversations in a crowded room with a series of microphones in the room recording the noise. In this problem, ICA would take the information recorded by the microphones and separate out the individual conversations around the room. Remember that these are distinct and different conversations going on in the same room, so one can assume that the signals (conversations) do not have any effect on each other.

Another classic example of ICA usage is in electroencephalographic (EEG) data. In both of these examples, there is a large amount of noise in the data. Noise is independent from the meaningful information, meaning that ICA is able to separate out the source of the noise. Knowing the source of the noise allows the user to bypass its influence on predictions (by

ignoring that data). Since GE data is also known for its noise, ICA has also been applied to it.

GE data can be handled by ICA in a very similar manner. Using our first example, instead of different conversations we have GO terms that are shared by groups of proteins, and the microphones in the example are replaced by the expression levels of the proteins in different experiments. For this, one can assume that the transcription levels of the proteins are independent. Sources in GE would be biologically significant groups of proteins [26].

Given a matrix X comprised of the original GE data, assume that there are some number of independent sources. The matrix is composed of microarray data, where there are m proteins with n experiments that they are subjected to. This results in an $m \times n$ matrix, which is the X matrix for ICA. Applying ICA to this matrix gives a vector S of sources and the $m \times n$ matrix A of linearly separable components of the biological processes.

In the past [7, 9, 26, 28, 30] ICA has been applied to GE data. For these experiments the input matrix is the GE data where rows are proteins and columns are experiments performed on those proteins. The resultant matrix can then be used to build a kernel, or have other algorithms applied to it (such as biclustering).

Using fastICA [33], I get projection matrix A, that divides the source information into independent components, and the vector S of source signals. I take this projection matrix and use it to generate a linear kernel.

3.3 Kernels for Sequence Data

I build linear kernels from sequence data. Each of these types of data are meant to emphasize certain areas of sequence similarity. Of these, only BLAST is considered to be standalone. The rest are supportive kernels, and should be used in conjunction with standalone kernels. When used alone they result in nearly random quality predictions.

3.3.0.1 BLAST

The Basic Local Alignment Search Tool (*BLAST*) is used to generate sequence alignment data. BLAST finds regions of sequences that are highly similar. It also calculates the statistical significance of the regions that it detects. For example, since some sequences are very common they are considered less important than an uncommon sequence that is found to be similar in two different protein sequences. BLAST data was post-processed by removing all hits with e-values less than 50.0, and dividing remaining values by 50.0 to normalize. The lowest allowed value was 1e-10, and values below that were changed to be that minimum value. The negative log of the resulting values were used as features.

3.3.1 Additional Sequence Based Kernels

Not all kernels are intended to make predictions on their own. Their purpose is to be combined with other kernels (for example BLAST and PPI based) to improve those kernels' predictions. These kernels highlight very specific information about proteins. They include the localization signals, transmembrane protein predictions, K-mer compositions of N and C termini, and low complexity regions in the sequence data.

While these kernels do not perform strongly on their own, when combined with other stronger kernels they can help to improve predictions. I can more confidently make predictions, since I am getting more information about the proteins that I am making predictions on. By having a larger pool of information to use, we are giving ourselves more clues to how those proteins are functioning. When combining kernels one can specify whether to use all samples or only those that are found across all data sets in the kernels being used.

Localization Biological processes are frequently localized to a certain area in the cell. By using this information, one can narrow down the possible protein functions for proteins in a locale. While there are a number of methods to extract this information, I used the WoLF PSORT algorithm [17]. This is an extension of PSORTII, a k-nearest-neighbor framework for localization information extraction [18]. I used the raw features computed by WoLF PSORT,

not its predictions.

Transmembrane Proteins are often embedded within the membrane of the cell and these proteins are often associated with certain functions, for example cell adhesion and transportation. Such proteins are inside of the membrane, inside the cell, and outside of the cell, and weave through these areas. The composition of the proteins changes depending on which area they are currently in. For each protein, I used the TMHMM [24] algorithm to determine how many transmembrane domains the protein has.

N and C Termini Proteins have 2 ends, which are known as the N and C termini. I took the 10 amino acids on each end and computed composition of the 3-mer representations of the N and C termini and used them as features.

Low Complexity Regions Low complexity regions in proteins are abundant, and often missed in standard protein sequence comparison methods. I used a sliding window of length 20 to scan these proteins and defined low complexity segments as those with the fewest distinct amino acids. The amino acid composition of the lowest complexity window was used as features.

3.4 Evaluating Results

Currently GOstruct uses a combination of PPI, GE, and sequence based data to make predictions. Some data sets are significantly more influential on the final predictions. First, not all of the data sets contain data on the same number or subset of proteins. Second, not all of the information in these data sets is equally important when determining GO terms applicability. In this second case, information pertaining to a certain area in the GO structure will not help much when making predictions in the other specialized areas. To avoid discriminating against data set size and composition I use a subset of gene names that appear in all data sets for each organism. While this does leave us with a smaller set of proteins to

Table 3.3: Existing evidence codes. The last evidence codes are ones that do not fit in the four groups. I used evidence codes that are not part of the ISS group.

Evidence Code Group	Ev. Code	Full Name
Experimental Evidence EXP	EXP	Inferred from Experiment
	IDA	Inferred from Direct Assay
	IPI	Inferred from Physical Interaction
	IMP	Inferred from Mutant Phenotype
	IGI	Inferred from Genetic Interaction
	IEP	Inferred from Expression Pattern
Comp. Analysis ISS	ISS	Inferred from Sequence or Structural Similarity
	ISO	Inferred from Sequence Orthology
	ISA	Inferred from Sequence
	ISM	Inferred from Sequence Model
	IGC	Inferred from Genomic Context
	RCA	Inferred from Reviewed Computational Analysis
Author Statement	TAS	Traceable Author Statement
	NAS	Non-traceable Author Statement
Curator Statement	IC	Inferred by Curator
	ND	No Biological Data Available
	IEA	Inferred from Electronic Annotation
	NR	Not Recorded

make GO predictions for, the change in data size is not significant enough to be of concern (for example, only a few *S. cerevisiae* terms are filtered out).

3.4.1 Filtering GO Evidence Codes

A large portion of annotations are produced computationally. The introduction of records created by bioinformaticists, that are created through automatic means and depend on existing (and sometimes erroneous) records, tend to propagate these errors to even more records [1]. Annotations in GO are assigned evidence codes that reflect their origin. Table 3.4.1 lists the current evidence codes, their group names, and their individual names.

Curated experimental data is considered of high quality [23]. Work has been done to determine the most reliable evidence codes. Jones estimates the error rates of curated records in the *GoSeqLite* database to be at approximately 28% [23]. When records with the ISS evidence codes are removed, the error rate drops to approximately 13% [23]. Rogers *et al.*

reinforced this in their 2009 paper, reiterating the ISS evidence codes are of lower reliability and should be removed [40]. Because of this, I also filter out ISS evidence codes for this work as a preproccessing step.

3.4.2 Calculating Error

In order to determine which of these kernel methods worked best, I had to first decide what would be construed as improvements.

One way to look at the error rate is on a per GO term basis. Instead of comparing the entire vector of GO term predictions for each protein, one can look at each GO term's correctness and use that to calculate the overall accuracy of the prediction. The area under the ROC curve (AUC) measures the rise of the false positive rate as the true positive rate increases. A high AUC indicates that there are significantly more true positives for every false positive, meaning that whatever method the AUC is measuring is generally correct. A low AUC indicates that there are many more false positives, showing that the measured method is often making incorrect predictions. The goal is to have a very high AUC score, meaning that the method is making many fewer incorrect predictions than correct predictions. Calculating AUC for structured output isn't trivial, so I do it on a GO term by GO term basis, which I believe makes it easier to track differences between methods. Another alternative is to use loss. I believe that it is easier to interpret the AUC values, so I use those to compare methods in this thesis.

3.4.3 Cross Validation

I evaluate the accuracy of the kernel methods in this thesis with five fold cross validation. When performing cross validation, the first step is to establish a set of data for each fold. In our problem if there are two similar proteins, each in its own fold, and a prediction is made of one based on the other, the prediction problem would be trivial. I prevent this from happening by ensuring that such proteins are within the same fold.

Folds are composed of groups of proteins with similar sequence. To do this, I create connected components of proteins on the basis of sequence similarity. I randomly assign

components to folds. The largest components are placed into the folds first. The smallest fold will be assigned the largest un-attached component, and this is repeated until all components are divided into the folds. This leads to fairly evenly sized folds where all similar proteins are contained in an individual fold. The same folds are used in all experiments. This maintains continuity across experiments; Regardless of which data sets are used, the same IDs will be used for training and testing models.

Chapter 4

Results and Discussion

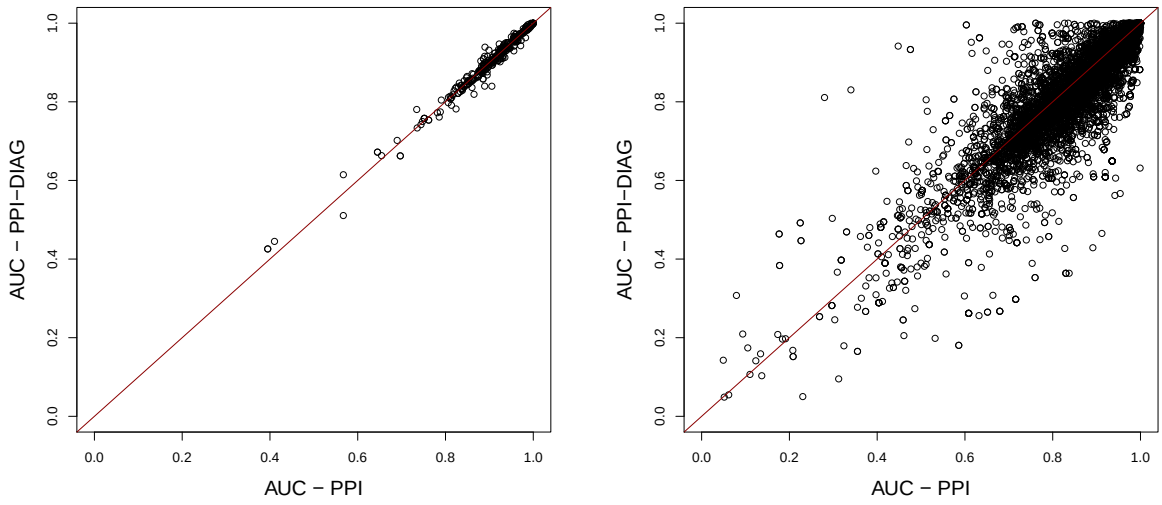
4.1 Network Data

I compare three kernel methods for the network data. The first, a linear kernel constructed from raw PPI data, is what we currently use with GOstruct [45, 46, 47]. The second and third, PPI-DIAG and the diffusion kernel, are new to GOstruct. PPI-DIAG is a completely new method, introduced in this work, and the diffusion kernel has been applied to PPI data in other instances [27, 49].

Table 4.1: Results from experiments over PPI data. Experiments were run on each GO namespaces– Molecular Function (MF), Biological Process (BP), and Cellular Component (CC), as well as the combined GO namespaces (MF, BP, and CC). The combined GO experiments make predictions on all of the GO namespaces simultaneously. Numbers are the average AUC values over the 5 folds of the of the data (which in turn are the averages of the AUC values for individual GO terms within the fold).

	PPI	PPI-DIAG	Diffusion
S. cerevisiae			
MF	0.952	0.900	0.486
BP	0.946	0.943	0.763
CC	0.946	0.944	0.744
Combined	0.955	0.953	0.583
M. musculus			
Combined	0.838	0.823	N/A

Table 4.1 shows the average AUC results from each of experiments using the kernels for network data. At first glance it appears that neither of the new methods (PPI-DIAG,



(a) *S. cerevisiae* (b) *M. musculus*

Figure 4.1: Scatter plots showing the AUC scores for GO terms, comparing the raw PPI kernel and PPI-DIAG. *S. cerevisiae* is shown in 4.1(a) and *M. musculus* is shown in 4.1(b).

diffusion) improve on the raw PPI results. In fact, the diffusion kernel performs much worse. For *S. cerevisiae*, the raw PPI kernel performs better than diffusion on approximately 1400 of the 1654 GO terms. For GO terms where the diffusion kernel performs better, the average AUC increase was only 0.12. PPI-DIAG results are very similar to the raw PPI kernel, so in Figure 4.1 I compare results for individual GO terms between the raw PPI kernel and PPI-DIAG. The two methods have a much higher degree of similarity between GO term AUC values in *S. cerevisiae* than in *M. musculus*. I believe this is because the interaction network in *S. cerevisiae* is better characterized than in *M. musculus*–there are more proteins and more interactions per protein.

Most GO terms are not strongly affected by changing the diagonal, but I believe that a certain subset of them will be. Of the approximately 7,000 GO terms predicted for *M. musculus*, PPI-DIAG has a higher AUC scores than the raw PPI kernel on 2569 terms. On another 4284 terms the raw PPI kernel does better, and the remaining few are exactly the same for both methods. GO terms that have higher AUC values in PPI-DIAG have an average AUC of 0.86, whereas average AUC for the same terms is 0.83 for the raw PPI

kernel.

4.2 Gene Expression

Each of the methods used in this work have had some success on the GE data. Initially I ran experiments on the individual kernel methods (which I will discuss shortly), and finding poor results, I experimented with combining kernels. Finally, I take the best performing experiments and combine those kernels with the sequence and network data.

Table 4.2: AUC values from experiments over GE data. These numbers are the average AUC values over the 5 folds of the data. Experiments were run on each GO namespaces–Molecular Function (MF), Biological Process (BP), and Cellular Component (CC), as well as the combined GO namespaces (MF, BP, and CC). The biclustering algorithms are the columns, Q1 meaning the QUBIC kernel that uses whole biclusters and Q2 meaning the kernel that separates biclustering into the normally regulated and oppositely regulated genes. 'ISA+Q2' is a combined run, where both the ISA and Q2 kernels were used. 'All GE' means that all of the GE kernels were used (Raw, ISA, Q1, Q2, and ICA).

	Raw	ISA	Q1	Q2	ICA	ISA+Q2	All GE
S. cerevisiae							
MF	0.641	0.587	0.582	0.604	0.633	0.608	0.671
BP	0.731	0.640	0.631	0.658	0.731	0.664	0.731
CC	0.718	0.656	0.619	0.634	0.759	0.666	0.728
Combined	0.752	0.650	0.630	0.650	0.759	0.666	0.765
M. musculus							
Combined	0.697	0.627	0.580	0.566	0.728	N/A	0.627

Table 4.2 shows AUC scores in 5-fold cross validation for the individual kernel experiments. These experiments were run on each GO namespace–Molecular Function (MF), Biological Process (BP), and Cellular Component (CC), as well the entire GO. For this, Q1 and Q2 signify the QUBIC kernels. 'ISA+Q2' is an experiment that used both the ISA and Q2 kernels. In the 'All GE' experiment, all of the GE kernels were used (Raw, ISA, Q1, Q2, and ICA). The results for each of the biclustering methods are very similar, and are all much lower than the results using the raw expression values. ICA performs slightly better than raw in most cases. Combining all of the GE kernels performs the best. These numbers are

averages over all of the GO terms in the experiments, so I take a closer look at the results to see if it makes sense to combine kernels.

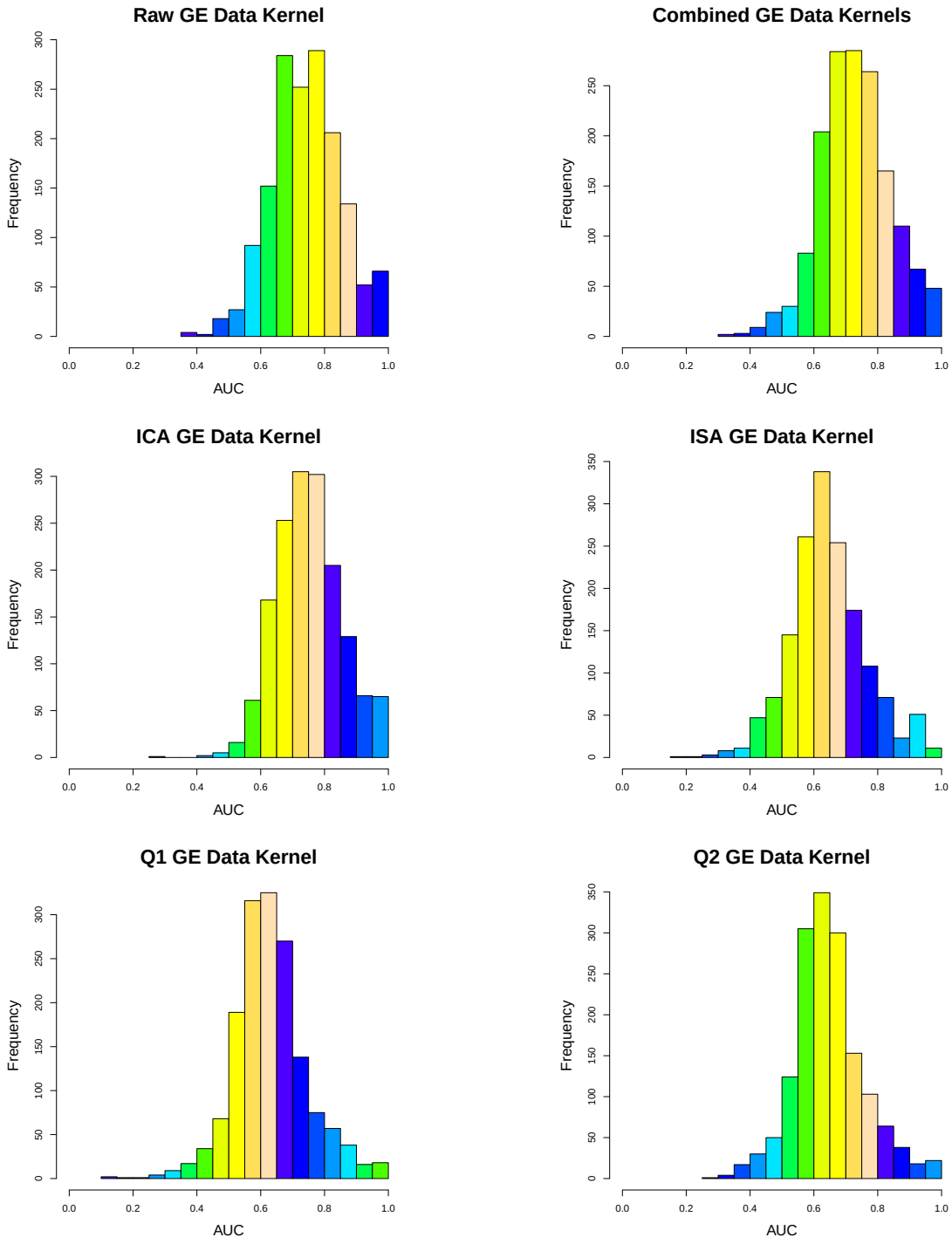


Figure 4.2: Distributions of AUC values for each of the kernels, and the results from the combined kernel experiments.

Figure 4.2 shows histograms of the AUC value distributions for each of the GE kernels. As expected from the averages in Table 4.2, the combined GE kernels experiment, ICA, and the kernel from the raw expression data all have more GO terms with AUC values above 0.5. ICA has a narrower distribution than the raw, and also has fewer extremely low scores. The biclustering algorithms are not as successful because they may not be finding comprehensive sets of biclusters, which is key for success in biclustering. Most terms have lower AUC values than their counterparts in the raw expression kernel experiment.

In binary prediction problems, I would not expect any terms to have AUC values below 0.5. Structured prediction, however, classifies a group of GO terms (in this thesis a group is either all GO terms, or all from a particular namespace) at a time, and may minimize the AUC value for one term so that it falls below 0.5 in order to maximize the scores of other GO terms. This is why we see some GO terms with AUC values below 0.5. The expectation is that by having those few GO terms with low scores, we will have many more with much higher scores.

The AUC for individual GO terms differed greatly between kernel methods. Figure 4.3 shows scatter plots comparing the GE kernel results. These plots show that the different methods are performing very differently on each GO term. The scatter plot comparing ICA and the raw expression data kernel shows that they are highly correlated. The biclustering scatter plots exhibit much less correlation, and have most points below the diagonal. A few points on the Q2 scatter plot have significantly higher AUC values for Q2 than for the raw kernel.

Figure 4.4 contrasts the biclustering algorithms. As stated before, these all performed poorly when compared to ICA and the kernel built from raw expression data. Comparing them allows us to see how much overlap there is between the different biclustering methods. From these scatter plots, we see that there is a large spread in the AUC values for individual GO terms between the biclustering methods. This indicates that the biclustering algorithms are indeed finding different important information.

To better visualize another key section of the output space, I look at all terms where the

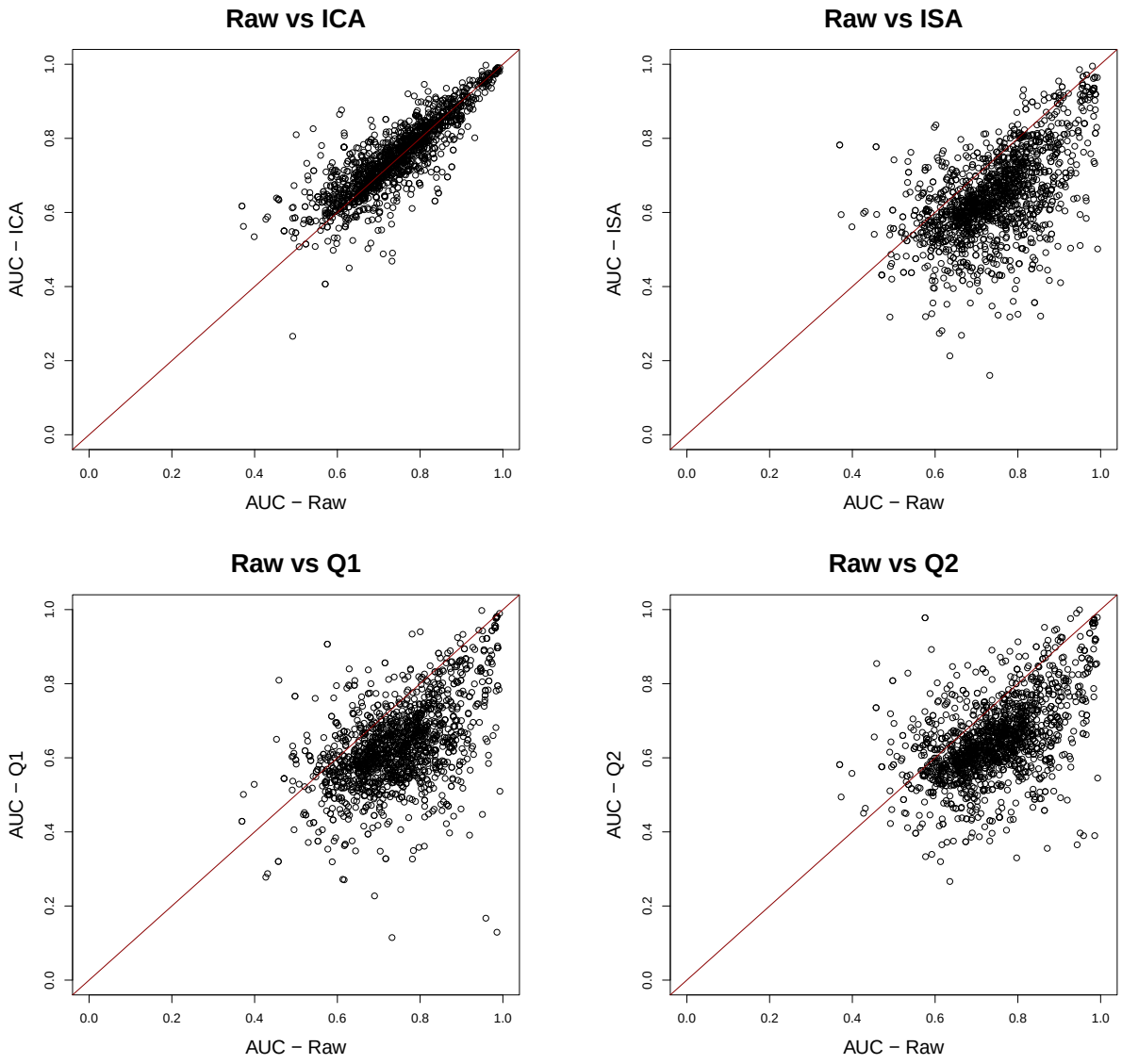


Figure 4.3: Scatter plots comparing the raw GE results to the biclustering methods and ICA.

kernel constructed from raw GE data performs the poorest. I compare the raw GE prediction results on these terms to the other methods. Figure 4.5 shows the 10 GO terms with the lowest AUC values for the raw GE data kernel. In this, each of the biclustering algorithms has significantly better results. ICA also does better than raw, but follows very closely in all but one case (GO:0005090, a molecular function term named *Guanine nucleotide exchange factor (GEF)*). This trend continues for many of the terms where the raw GE data does not predict well. The question becomes this: what causes this, and is there an optimal combination of

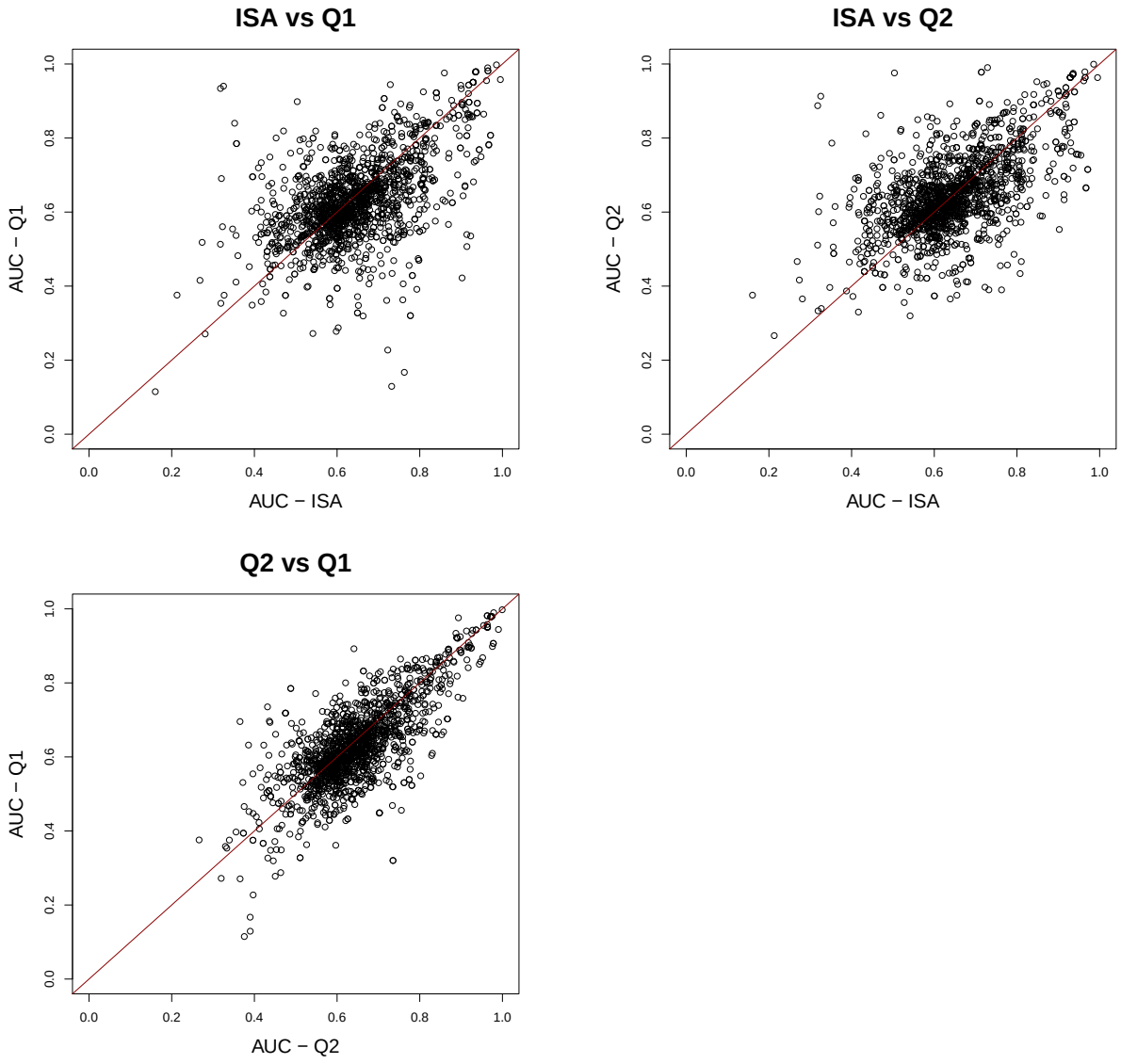


Figure 4.4: Scatter plots comparing each of the biclustering methods to the others. The two forms of QUBIC are present, as well as ISA.

kernels to capture key information?

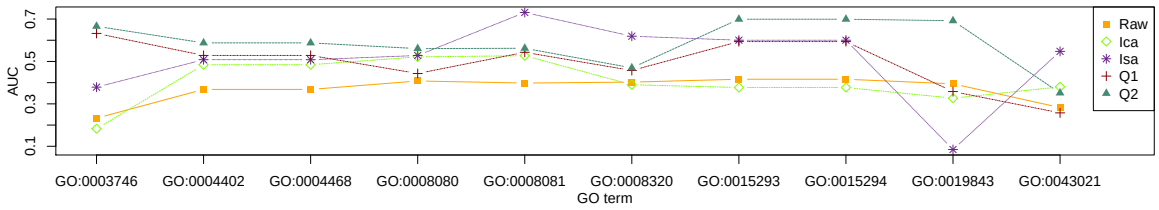


Figure 4.5: The 10 lowest scored GO terms for the raw GE data (all have less than 0.5 AUC), and their AUC for each of the biclustering methods. This is over the molecular function namespace using the 2011 GE *S. cerevisiae* data. All of the biclustering methods except for ICA perform significantly better on these terms.

Figure 4.5 shows that for all instances where the raw GE kernel performs poorly, the biclustering algorithms are highly successful. This suggests that the biclustering kernels are taking advantage of different significant information in the GE data than the raw expression kernel, implying that the methods complement each other. Because of this, when combining kernels, I decided to include the biclustering kernels with other GE kernels.

Figure 4.6 takes another approach in viewing the results from the individual kernel experiments. Here, I take the two GO terms from each GE kernel experiment that have the highest scores. There are 5 methods (raw, ICA, ISA, Q1, and Q2), so we expect to end up with 10 GO terms. Overlap between the highest terms in the different kernels led us to have fewer than 10 terms in the final figure. In this figure we see a strong correlation between results in almost all terms. GO:0031428 (*box C/D snoRNP complex*) from the cellular component namespace has the most variation, with AUC scores ranging from almost 0 (Q1) to almost 1 (ICA and raw).

Table 4.2 shows the combined kernel experiment results. ICA and raw were the top performing GE kernels, so I combined each of them with the sequence and PPI data. I also used ISA and one of the QUBIC kernels (Q2) to represent the GE data, in another set of tests. Finally, I use all of the GE kernels as well as the sequence and PPI data. These results didn't show improvements using GE data. There are a couple of reasons why this may be. The PPI data for *S. cerevisiae* is very accurate and complete, and in these cases may be

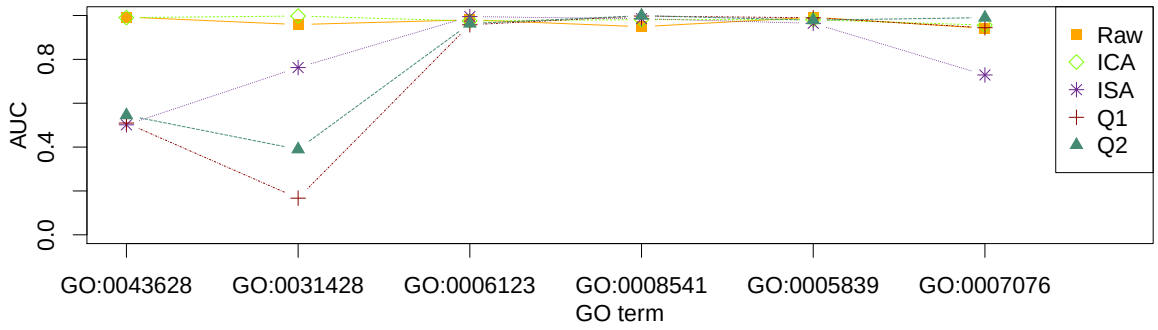


Figure 4.6: The two best terms (in the combined GO namespace) for each of the kernels to predict, for the *S. cerevisiae* experiments. I plot the AUC values for all kernels, for the list of terms that were the best from each kernel.

Table 4.3: Results from expression data experiments over the *S. cerevisiae* data with combinations of GE kernels. Experiments were run on each GO namespace. In this table, "All" means the sequence and PPI kernels. The GE kernels that were used are listed in the column names. These experiments use different GE kernels in conjunction with the PPI and sequence data kernels. 'All but GE' is the baseline experiment, and does not use any GE kernels.

	All but GE	All+Raw	All+ISA+Q2	All+ICA
MF	0.947	0.947	0.941	0.946
BP	0.941	0.940	0.935	0.939
CC	0.953	0.947	0.940	0.939
Combined	0.953	0.953	0.949	0.953

overwhelming the improvements from other data. The results in Table 4.2 should be taken with the understanding that this is the case, and that improving GE predictions is valuable.

4.3 Sequence Data

I had several sequence kernels and were interested in seeing the contribution from each. Table 4.2 shows results from the sequence based kernels and the original PPI kernel (*S. cerevisiae* data for only the molecular function namespace, not the combined GO namespace). This table illustrates how, alone, many of the sequence based kernels are only marginally better than random predictions. However, when combined with other stronger kernels they help to improve predictions. For our experiments I used these kernels in conjunction with the network and GE data. Each of the sequence based kernels does not perform well alone,

Table 4.4: Results from existing data sources. This is AUC for all molecular function GO terms using the *S. cerevisiae* data. In this table, 'seq' means all of the sequence data kernels (BLAST, lcomplx, localiz, termini, and tramsmem), and 'seq+network' means all of the sequence data kernels as well as the raw PPI kernel.

	Avg
Network Kernels	
PPI	0.952
Sequence Kernels	
blast	0.700
lcomplx	0.702
localiz	0.635
termini	0.552
transmem	0.566
Combined Kernels	
seq+network	**0.956**

however, combined they do much better.The 'seq+network' experiment used the PPI kernel as well as all of the sequence based kernels (BLAST, lcomplx, localiz, termini, and transmem). Because the network data is so well characterized in *S. cerevisiae*, the combined experiment has a very high AUC average. Combining the sequence kernels with the network kernels does provide some improvement.

Chapter 5

Conclusions and Future Work

Three kernel methods were used on the network data–PPI, PPI-DIAG, and the diffusion kernel. Of these methods, the original linear kernel constructed from adjacency data was most successful overall. Based on these results, I would recommend that users continue to use the linear kernel on PPI data. Unlike the supportive kernels discussed in Section 3.3, the network kernels are not suited to being combined, because they have a high degree of similarity in the results. The diffusion kernel has been shown to be applicable to other network problems [27, 49], and PPI-DIAG may similarly have other applications where it would be successful.

While other works [27, 49] have found instances where the diffusion kernel is effective, it was not the case here. The data used by other works is also different from that used in this thesis. In these cases, the data had a binary format (either the proteins interact or they do not), rather than being confidence levels. Furthermore I am using a structured output SVM instead of a series of binary SVMs. The diffusion kernel may not be effective with structured output methods. I am not sure which of these affect the diffusion kernel results but it is clear that in this instance the diffusion kernel is not an improvement over the kernel I currently use (the linear PPI kernel).

Gene expression data has proved difficult to make effective use of in machine learning algorithms because it is highly noisy. I explored several methods for extracting robust features from this data– several forms of biclustering, and ICA. I have found that different biclustering methods find vastly different relationship. So by combining these different viewpoints we

are able to extract a more comprehensive view of the data, which translates into better performance.

Of all the GE kernels, only ICA outperforms the raw kernel and even it is unable to improve the overall results when combined with the sequence and network data. Table 4.2 shows results from the ICA experiments and it outperforms raw in almost all cases (the MF namespace for *S. cerevisiae* being the exception). This may not be the case in other species, for example *M. musculus* and *Homo sapiens* (human), where the methods developed here could be more useful. Network data for *M. musculus* and *H. sapiens* is not as effective as the *S. cerevisiae* network data, and so the GE results would have more opportunity to make improvements in predictions. The results in Figure 4.2 should be taken with the understanding that the network data overshadows the other data, and that improving GE predictions may be valuable in other organisms.

Our preliminary investigations in combining kernels, shown in Table 4.2, show promising improvements. The biclustering and ICA kernels improve overall performance, and I recommend that users take advantage of these kernels. This doesn't translate into improvement in experiments using sequence and network data. I believe that this is because in *S. cerevisiae* the network data is very well characterized. I would like to continue investigation in *M. musculus* and *H. sapiens* because the GE only experiments in *S. cerevisiae* show potential to improve the combined (sequence, network, and GE data) experiment results and in *M. musculus* and *H. sapiens* the network data is not as well characterized. Continuing investigation into these approaches should lead to the best combination of kernels, allowing us to remove others from use, minimizing overhead.

Improvements are most noticeable in the Biological Process (BP) and Cellular Component (CC) namespaces, but are present in Molecular Function (MF) as well. Furthermore, when predicting across the entire GO namespace these improvements do not deteriorate. I intend to continue making predictions using the combined GO namespaces.

As mentioned above, I have plans for several extensions to this work. Experiments take too long to run; I need faster methods. As discussed in Chapter 4, run times for experiments

Table 5.1: Approximate system requirements for *S. cerevisiae*, *M. musculus*, and *H. sapiens* experiments. These tests are run on 64 bit Fedora Core 16 machines with 16Gb memory, 8 core 3.0G processors.

	Run Time (days)
S. cerevisiae	1
M. musculus	14
H. sapiens	28

can take several weeks (see Table 5). The memory requirements for such tests are also extensive. I would like to see work done on making GOstruct utilize parallel processing more than it does now. Once the memory and runtime issues have been solved, I want to apply these methods to a larger model organism–*Homo sapiens*. I believe that *H. sapiens* would provide additional insight to these kernel methods.

REFERENCES

[1] Carson Andorf, Drena Dobbs, and Vasant Honavar. Exploring inconsistencies in genome-wide protein function annotations: a machine learning approach. *BMC Bioinformatics*, 8(1):284, 2007.

[2] Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830, 2006.

[3] Asa Ben-Hur, Cheng S. Ong, Sören Sonnenburg, Bernhard Schölkopf, and Gunnar Rätsch. Support Vector Machines and Kernels for Computational Biology. *PLoS Comput Biol*, 4(10), October 2008.

[4] P. Bork and E. V. Koonin. Predicting functions from protein sequences - where are the bottlenecks? *Nature Genetics*, 18:313–318, 1998.

[5] Corinna Cortes and Vladimir Vapnik. Support-Vector Networks. In *Machine Learning*, volume 20, pages 273–297, 1995.

[6] D. Devos and A. Valencia. Practical limits of function prediction. *PROTEINS-NEW YORK-*, 41(1):98–107, 2000.

[7] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, December 1998.

[8] B.E. Engelhardt, M.I. Jordan, K.E. Muratore, and S.E. Brenner. Protein molecular function prediction by Bayesian phylogenomics. *PLoS computational biology*, 1(5):e45, 2005.

[9] Jesse M. Engreitz, Bernie J. Daigle Jr, Jonathan J. Marshall, and Russ B. Altman. Independent component analysis: mining microarray data for fundamental human gene expression modules. *Journal of biomedical informatics*, July 2010.

[10] M. Y. Galperin and E. V. Koonin. Sources of systematic error in functional annotation of genomes: domain rearrangement, non-orthologous gene displacement, and operon disruption. *In silico Biology*, 1:55–67, 1998.

[11] Carole Goble and Robert Stevens. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41(5):687 – 693, 2008. Semantic Mashup of Biomedical Data.

[12] Vincent Goulet, Christophe Dutang, Martin Maechler, David Firth, Marina Shapira, Michael Stadelmann, and expm-developers@lists.R-forge.R-project.org. *expm: Matrix exponential*, 2011. R package version 0.98-4.

[13] Casey S. Greene and Olga G. Troyanskaya. PILGRIM: an interactive data-driven discovery platform for expert biologists. *Nucleic acids research*, 39(Web Server issue):W368–W374, July 2011.

[14] T. Hamp, R. Kassner, S. Seemayer, E. Vicedo, et al. Nearest-neighbor approaches to predict protein function by homology inference alone. In *Automatic Function Prediction special interest group meeting at ISMB*, 2011.

[15] S. Hennig, D. Groth, and H. Lehrach. Automated gene ontology annotation for anonymous sequence data. *Nucleic Acids Research*, 31(13):3712, 2003.

[16] Sepp Hochreiter, Ulrich Bodenhofer, Martin Heusel, Andreas Mayr, Andreas Mitterecker, Adetayo Kasim, Tatsiana Khamiakova, Suzy Van Sanden, Dan Lin, Willem Talloen, Luc Bijnens, Hinrich W. H. Ghlmann, Ziv Shkedy, and Djork-Arn Clevert. Fabia: factor analysis for bicluster acquisition. *Bioinformatics*, 26(12):1520–1527, 2010.

[17] Paul Horton, Keun-Joon Park, Takeshi Obayashi, Naoya Fujita, Hajime Harada, C. J. Adams-Collier, Kenta Nakai, and Kenta Nakai. Wolf psort: protein localization predictor. pages 585–587, 2007.

[18] Paul Horton, Keun-Joon Park, Takeshi Obayashi, and Kenta Nakai. Protein subcellular localisation prediction with wolf psort. In *APBC*, pages 39–48, 2006.

[19] J. Ihmels, G. Friedlander, S. Bergmann, O. Sarig, Y. Ziv, and N. Barkai. Revealing modular organization in the yeast transcriptional network. *Nat Genet*, 31(4):370–377, August 2002.

[20] Jan Ihmels, Sven Bergmann, and Naama Barkai. Defining transcription modules using large-scale gene expression data. *Bioinformatics*, 20:2004, 2004.

[21] L.J. Jensen, M. Kuhn, M. Stark, S. Chaffron, C. Creevey, J. Muller, T. Doerks, P. Julien, A. Roth, M. Simonovic, et al. STRING 8.a global view on proteins and their functional interactions in 630 organisms. *Nucleic acids research*, 37(suppl 1):D412, 2009.

[22] T. Joachims. Making large-scale svm learning practical. In *Advances in kernel methods: support vector learning*, pages 169–184. MIT Press, 1999.

[23] Craig Jones, Alfred Brown, and Ute Baumann. Estimating the annotation error rate of curated go database sequence annotations. *BMC Bioinformatics*, 8(1):170, 2007.

[24] Robel Y. Kahsay, Guang R. Gao, Li Liao, and Li Liao. An improved hidden markov model for transmembrane protein detection and topology prediction and its applications to complete genomes. pages 1853–1858, 2005.

[25] R.I. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 315–322, 2002.

[26] Wei Kong, Charles R. Vanderburg, Hiromi Gunshin, Jack T. Rogers, and Xudong Huang. A review of independent component analysis application to microarray gene expression data. *Biotechniques*, 45(5), 2008.

[27] H. Lee, Z. Tu, M. Deng, F. Sun, and T. Chen. Diffusion kernel-based logistic regression models for protein function prediction. *OMICS: A Journal of Integrative Biology*, 10(1):40–55, 2006.

[28] S. I. Lee and S. Batzoglou. Application of independent component analysis to microarrays. *Genome Biol*, 4(11), 2003.

[29] Guojun Li, Qin Ma, Haibao Tang, Andrew H. Paterson, and Ying Xu. QUBIC: a qualitative biclustering algorithm for analyses of gene expression data. *Nucleic acids research*, 37(15), August 2009.

[30] Wolfram Liebermeister. Linear modes of gene expression determined by independent component analysis. *Bioinformatics*, 18(1):51–60, January 2002.

[31] Y. Loewenstein, D. Raimondo, O. Redfern, J. Watson, D. Frishman, M. Linial, C. Orengo, J. Thornton, and A. Tramontano. Protein function annotation by homology-based inference. *Genome Biology*, 10(2):207, 2009.

[32] Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, Davis AP, Dolinski K, Dwight SS, Eppig JT, Harris MA, Hill DP, Issel-Tarver L, Kasarskis A, Lewis S, Matese JC, Richardson JE, Ringwald M, Rubin GM, and Sherlock G. Gene ontology: tool for the unification of biology. the gene ontology consortium. *Nature Genetics*, 25(1):25–29, May 2000.

[33] J L Marchini, C Heaton, and B D Ripley. *fastICA: FastICA Algorithms to perform ICA and Projection Pursuit*, 2010. R package version 1.1-13.

[34] D. Martin, M. Berriman, and G. Barton. Gotcha: a new method for prediction of protein function assessed by the annotation of seven genomes. *BMC bioinformatics*, 5(1):178, 2004.

[35] J. McDermott, R. Bumgarner, and R. Samudrala. Functional annotation from predicted protein interaction networks. *Bioinformatics*, 21(15):3217, 2005.

[36] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20:801–836, 1978.

[37] S. Mostafavi, D. Ray, D. Warde-Farley, C. Grouios, and Q. Morris. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. *Genome Biology*, 9(Suppl 1):S4, 2008.

[38] G. Obozinski, G. Lanckriet, C. Grant, M. Jordan, and W. Noble. Consistent probabilistic outputs for protein function prediction. *Genome Biology*, 9(Suppl 1):S6, 2008.

[39] L. Peña-Castillo, M. Tasan, C. Myers, H. Lee, T. Joshi, C. Zhang, Y. Guan, M. Leone, A. Pagnani, W. Kim, et al. A critical assessment of *Mus musculus* gene function prediction using integrated genomic evidence. *Genome Biology*, 9(Suppl 1):S2, 2008.

[40] Mark F. Rogers and Asa Ben-Hur. The use of gene ontology evidence codes in preventing classifier assessment bias. *Bioinformatics*, 25(9):1173–1177, 2009.

[41] B. Rost, J. Liu, R. Nair, K.O. Wrzeszczynski, and Y. Ofran. Automatic prediction of protein function. *Cellular and Molecular Life Sciences*, 60(12):2637–2650, 2003.

[42] M. Schena, D. Shalon, R.W. Davis, and P.O. Brown. Quantitative monitoring of gene expression patterns with a complementary dna microarray. *Science*, 270(5235):467, 1995.

[43] J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge Univ Pr, 2004.

[44] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA, 2004.

[45] A. Sokolov and A. Ben-Hur. Multi-view prediction of protein function. In *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, 2011.

[46] Artem Sokolov and Asa Ben-Hur. Hierarchical classification of gene ontology terms using the gostruct method. *J Bioinform Comput Biol*, 8(2):357–76, 2010.

[47] Artem Sokolov, Chris Funk, Kiley Graim, Karin Verspoor, and Asa Ben-Hur. Combining heterogeneous data sources for accurate functional annotation of proteins. *BMC Bioinformaticsi*, Submitted 2012.

[48] 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(Database Issue):D535, 2006.

[49] Naifang Su, Lin Wang, Yufu Wang, Minping Qian, Minghua Deng, and A Materials. Prediction of protein functions from protein-protein interaction data based on a new measure of network betweenness. *Bioinformatics and Biomedical Engineering iCBBE 2010 4th International Conference on*, (Mcl):1–4, 2010.

[50] A. Tanay, R. Sharan, M. Kupiec, and R. Shamir. Revealing modularity and organization in the yeast molecular network by integrated analysis of highly heterogeneous genomewide data. *Proc Natl Acad Sci USA*, 101(9):2981–6, 2004.

[51] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453, 2006.

[52] A. Vazquez, A. Flammini, A. Maritan, and A. Vespignani. Global protein function prediction in protein-protein interaction networks. *Nature Biotechnology*, 21(6):697–700, 2003.

[53] Christian von Mering, Lars Juhl Jensen, Berend Snel, Sean D. Hooper, Markus Krupp, Mathilde Foglierini, Nelly Jouffre, Martijn A. Huynen, Peer Bork, and Peer Bork. String: known and predicted protein-protein associations, integrated and transferred across organisms. pages 433–437, 2005.

[54] G. Xiao and W. Pan. Gene function prediction by a combined analysis of gene expression data and protein-protein interaction data. *Journal of bioinformatics and computational biology*, 3(6):1371, 2005.

[55] G. Zehetner. Ontoblast function: From sequence similarities directly to potential functional annotations by ontology terms. *Nucleic acids research*, 31(13):3799, 2003.