Computational Prediction of Gene Function from High-throughput Data Sources

by

Sara Mostafavi

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Computational Prediction of Gene Function from High-throughput Data Sources

Sara Mostafavi

Doctor of Philosophy

Graduate Department of Computer Science

University of Toronto

2011

A large number and variety of genome-wide genomics and proteomics datasets are now available for model organisms. Each dataset on its own presents a distinct but noisy view of cellular state. However, collectively, these datasets embody a more comprehensive view of cell function. This motivates the prediction of function for uncharacterized genes by combining multiple datasets, in order to exploit the associations between such genes and genes of known function–all in a query-specific fashion.

Commonly, heterogeneous datasets are represented as networks in order to facilitate their combination. Here, I show that it is possible to accurately predict gene function in seconds by combining multiple large-scale networks. This facilitates function prediction on-demand, allowing users to take advantage of the persistent improvement and proliferation of genomics and proteomics datasets and continuously make up-to-date predictions for large genomes such as humans.

Our algorithm, GeneMANIA, uses constrained linear regression to combine multiple association networks and uses label propagation to make predictions from the combined network. I introduce extensions that result in improved predictions when the number of labeled examples for training is limited, or when an ontological structure describing a hierarchy of gene function categorization scheme is available. Further, motivated by our empirical observations on predicting node labels for general networks, I propose a new label propagation algorithm that exploits common properties of real-world networks to increase both the speed and accuracy of our predictions.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

An unrealized goal of computational molecular biology is to determine the functional roles of all genes (or proteins) in a cell. Even in a well-studied organism like *Saccharomyces cerevisiae* (budding yeast), the precise function[1] of 15% of the gene complement (as of April 2010) remains unknown. Initial computational attempts for discerning function have primarily relied on sequence similarity (homology) to infer the functions of uncharacterized genes, based on the functions of their homologs. However, this method on its own has a number of limitations. Sequence similarity is often more informative of molecular function, rather than higher-level biological roles. Further, evolutionary processes like domain shuffling and gene duplication result in homologs that may not have the same function.

The past decade has seen a large increase in the quantity and variety of publicly available genomic and proteomic data, aside from sequence information. These genome-wide data, which encompass gene expression, protein and genetic interaction networks, phylogenetic profiles, and domain composition, are available for a large proportion of genes in a given genome. Each data type captures distinct gene features that provide indirect information about functional roles of uncharacterized genes. This type of indirect infor-

[1]We define a *precise* function as a function that involves 500 or fewer genes.

mation can be refined into direct predictions by associating uncharacterized genes with those with known functions. For instance, genes with similar expression patterns [11], genetic interaction profiles [106, 93], domain composition [36], or phylogenetic profiles [75] tend to have similar functions. As well, function tends to be shared among genes whose gene products interact physically [98], or are part of the same complex [102]. Collectively, these observations have led to functional categorization of a number of previously uncharacterized genes using the so called "guilt-by-association" principle.

However, none of these genome-wide data are comprehensive on their own, and more accurate predictions can be made by combining multiple data types [48, 97, 74, 95, 56]. A common preliminary step for combining these different data types is to first represent each as an affinity network (also referred to as functional linkage network or functional association network or functional interactions). In these networks, nodes represent genes and the edges represent the strength of evidence for the co-functionality between the connected genes, as derived from feature similarities according to a given dataset. Most generally, the problem that we address in this thesis can be stated as follows: given multiple networks, each constructed from a genomics or proteomics dataset, and a list of positive genes (those known to be involved in a given function), predict other genes that are likely to be positives. In machine learning, this problem can be formulated as binary classification in a transductive setting where both training data (positive genes) and test data (uncharacterized genes) are available at training time.

Previous approaches to inferring gene functions from affinity networks fall under two categories: those that make few assumptions and attempt to solve a challenging optimization problem [48, 97] and those that simplify the task by making several limiting assumptions [63, 64, 102] like independence between the different networks or only consider direct network neighbors to transfer function between genes. In practice, new experimentally-derived function assignments and genome-wide datasets are being generated at a rapid rate. To take advantage of these frequently updated data, it is important to design algorithms that can be implemented on “prediction servers”, allowing users to define positive gene lists and use arbitrary combinations of networks to make “on-demand” function predictions, as opposed to static databases that attempt to pre-compute a large number of pre-defined queries. This motivates a design that scales to hundreds of networks with tens of thousands of genes (nodes). However, the minimal-assumption approach is not scalable for on-demand predictions, and the simplified approach may be too restrictive and is often not robust enough to be used with arbitrary numbers and varieties of networks.

Therefore, the focus of this thesis is on developing fast yet accurate algorithms for predicting gene function from multiple networks. We study three related tasks: 1) how to efficiently combine multiple networks for a given prediction task while accounting for redundant or irrelevant datasets, 2) how to make predictions from the combined network, and 3) how to incorporate auxiliary information about gene functions, such as the hierarchical organization of gene function categories, when making predictions. We hypothesize that by formulating well-posed convex optimization tasks, we can develop efficient algorithms that nonetheless make only appropriate assumptions. In particular, we propose methods that take advantage of the strengths of both of the above approaches; as we will show, doing so results in both improved performance and efficiency.

More concretely, we introduce a linear regression approach for combining multiple networks on a query-specific basis and use a version of the label propagation algorithm [109, 107] to predict gene function from a combined network. This work is encapsulated in the GeneMANIA framework. Further, we propose extensions to this framework to improve network integration when only a small amount of labeled data is available, and to incorporate information about the hierarchical organization of function categories. Finally, we propose an alternative to label propagation algorithm that results in faster and more accurate predictions. Applying these methods to several model organisms, we show that we can simultaneously equal or exceed the accuracy and efficiency of the existing methods. The work in this thesis encompasses the following publications:

- Mostafavi et al. 2008. GeneMANIA: A real-time multiple functional association network integration algorithm for predicting gene function. *Genome Biology.* (Suppl. 9):S2.
- Mostafavi and Morris. 2009. Incorporating the structure of gene ontology hierarchy when predicting gene function. *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI).* Montreal, QC.
- Mostafavi and Morris. 2010. Fast integration of heterogeneous data sources for predicting gene function with limited annotation. *Bioinformatics.* 26:1759–1765.
- Costanzo, Baryshnikova, et al. 2010. The genetic landscape of a cell. *Science.* 327:425–431.
- Mostafavi, Goldenberg, and Morris. 2010. Predicting node characteristics from molecular networks. *Methods in Molecular Biology: Network Biology.* (In review).
- Mostafavi, Goldenberg, and Morris. 2010. Three degrees of propagation. (Under preparation).

Thesis Outline

The content of this thesis is organized as follows:

- **Chapter 2**: *Background.* In this chapter, we provide background on types of gene function, varieties of genome-wide datasets and functional linkage networks, and existing approaches to the three tasks that we consider in this thesis: predicting gene function from a single network, predicting gene function from multiple networks, and incorporating ontology structure when making predictions. As well, we describe standard evaluation metrics for assessing the performance of gene function prediction.

- **Chapter 3**: *Predicting Gene Function from Multiple Networks in Seconds.* In this chapter, we present the GeneMANIA framework, which consists of a constrained linear regression algorithm for combining multiple networks, and a label propagation algorithm for predicting gene function from the combined network. We present experimental results on combining a large collection of networks in several example organisms and show that GeneMANIA outperforms previous approaches.

- **Chapter 4**: *Incorporating Ontology Structure into Predictions.* In this chapter, we investigate the task of incorporating hierarchical ontology structure into predictions. We propose two new methods for incorporating ontology information based on label propagation. Experiments show that our methods greatly improve the performance of gene function prediction.

- **Chapter 5**: *Predicting Binary Node Labels in Very Large Networks.* In this chapter, we generalize our framework for inference of node labels from functional linkage networks and consider the general problem of predicting binary node labels from an arbitrary network. Motivated by our empirical results for iterative label propagation, we propose a new approach, *weighted proximal propagation (WPA)*, that exploits empirical properties of real-world networks to make faster and more accurate predictions. Experimental results on several network types show the advantages of the proposed approach.

- **Chapter 6**: *Conclusions and Future Work.* In this chapter, we discuss future work and summarize the contributions of this thesis.

Chapter 2

Background

In this chapter we provide background on the three principal tasks that we consider in this thesis: predicting gene function from a single network, predicting gene function from multiple networks, and incorporating ontology structure when making predictions. Before doing so, we define the precise problem setting and relevant terminology. First, we describe how gene functions are defined and discuss current practices in associating genes with functions. Then we describe genomics and proteomics datasets that are commonly used in the context of guilt-by-association to infer the functions of uncharacterized genes. Next, we describe how to represent these heterogeneous genomics and proteomics data sources using networks, and describe relevant network definitions and terminology. Finally, after describing the existing work on these three aspects of gene function prediction, we describe standard evaluation metrics for evaluating the performance of a function prediction method.

2.1 Describing Gene Functions

To predict gene function, we use as training data a list of genes currently known to have a function of interest. We now describe how gene functions are defined and organized.

There are several online databases, such as Gene Ontology (GO) [18], Kyoto Ency-

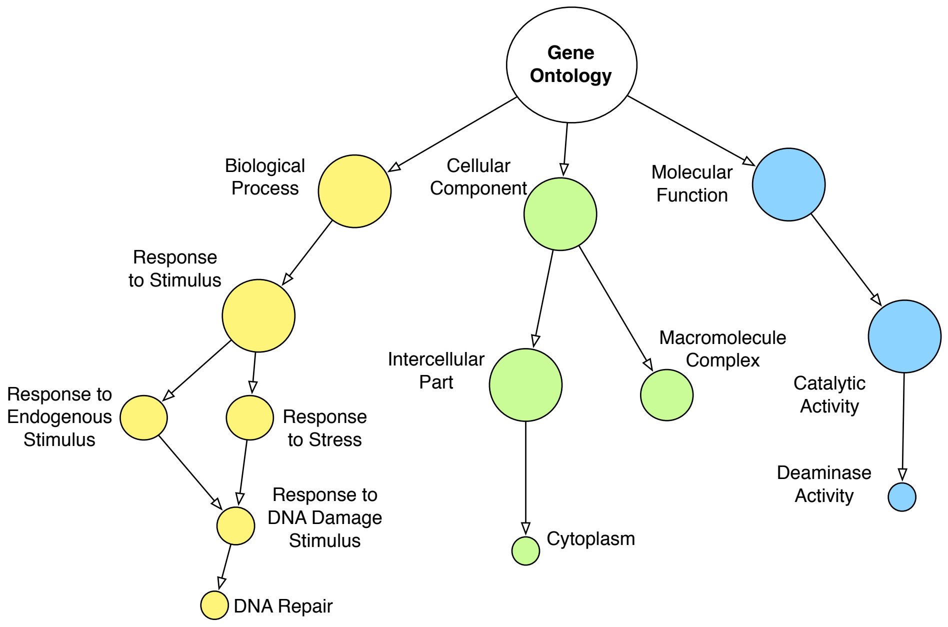


Figure 2.1: A graphical example of the organization of Gene Ontology (GO). This figure shows the three hierarchies defined by GO (Biological Process, Cellular Components, and Molecular Function) as well as example functions in each hierarchy.

clopedia of Genes and Genomes (KEGG) [43], and Enzyme Commission (EC) [4], that both define a controlled vocabulary for describing gene function and provide known lists of genes associated with each function. GO is one of the most widely used ontologies; it is organism-independent, and as of 2010 it defines tens of thousands of functions (referred to as GO terms). Gene Ontology defines three hierarchies of GO terms: Biological Processes (BP), Cellular Components (CC), and Molecular Functions (MF). Each hierarchy organizes GO terms into a directed acyclic graph where the leaves represent most specific functions and nodes near the root represent the broadest types of function (see Figure 2.1). Further, annotations (defined as the assignment of a gene to a GO term) satisfy the “true path” rule: genes annotated to a given category are also assigned to all of its ancestors.

In GO, each annotation is associated with an evidence code. Broadly, there are five types of evidence codes: annotations based on experimental evidence, reviewed computational analysis, unreviewed computational analysis, author statement, and curator statement (see Table 2.1). The unreviewed computational evidence code (called Inferred from Electronic Annotation (IEA)), is by far the most prevalent type of function annotation; for instance, as of October 2010, 50% of GO function annotations in human are based on IEA evidence alone. When training a classifier, it is standard practice to discard annotations based on such evidence [76]. This is because such annotations are themselves computational predictions that have not been reviewed by a curator or published elsewhere, and their inclusion results in an over-estimate of performance [82, 76]. In our experiments, we follow the same practice and discard IEA annotations. On the other hand, only a small fraction of function annotations are derived from computational methods that combine multiple data sources (RCA evidence code); for instance, 0.01% of annotation of human genes have an RCA evidence code. This further motivates the need for scalable computational approaches for making predictions from multiple high-throughput data sources.

2.2 Describing Genome-Wide Data

We now consider the types of genome-wide data that are used to predict gene function. Traditionally, protein sequence data has served as one of the most widely used sources for predicting shared functionality. In recent years, other commonly available data types include gene expression, protein and genetic interaction networks, protein localization, and domain composition data. Much of this data is available from online databases such as BioGRID [91], Gene Expression Omnibus (GEO) [24], Pfam [28], and BioMart [87]. Below, we briefly review these common data types (see Table 2.2 for a summary).

Protein Sequence: Protein sequences are strings of variable lengths constructed from amino acid molecules. There are twenty standard amino acids that constitute most proteins in nature. Similarity between protein sequences indicate their evolutionary relationship, and sequence similarity is informative of shared molecular function.

Table 2.1: List of Gene Ontology evidence codes. Each evidence code is categorized into one of five broader categories: Experimental Evidence, Computational Analysis, Author Statement, Curator Statement, and Automatically Assigned. The first four categories (Experimental Evidence, Computational Analysis, Author Statement, and Curator Statement) are reviewed by GO annotators whereas the Automatically Assigned category is unreviewed by the annotators.

Experimental Evidence Codes
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

Reviewed Computational Analysis Evidence Codes
ISS: Inferred from Sequence or Structural Similarity
ISO: Inferred from Sequence Orthology
ISA: Inferred from Sequence Alignment
ISM: Inferred from Sequence Model
IGC: Inferred from Genomic Context
RCA: Inferred from Reviewed Computational Analysis

Author Statement Evidence Codes
TAS: Traceable Author Statement
NAS: Non-traceable Author Statement

Curator Statement Evidence Codes
IC: Inferred by Curator
ND: No biological Data available

Automatically Assigned Evidence Codes
IEA: Inferred from Electronic Annotation

Protein Domains and Motifs: Proteins sequences can be decomposed into functional subunits known as domains. Each domain can be independently folded. For example, zinc finger domain is a common domain in DNA-binding proteins such as transcription factors. Motifs are shorter sequences that are found in multiple proteins. For instance, there are known motifs that encode signaling peptides (e.g. available from SignalP [27]). Other examples include motifs that encode cellular localization, such as those that are predictive of cytoplasmic, nuclear, secreted, or transmembrane proteins. In general, proteins that are comprised of similar domains or motifs may have similar functions [42]. Domain (or sequence motif) data is often represented in an $n \times d$ matrix where n is the number of genes and d is the number of distinct domains. The $(i, j)^{\text{th}}$ entry in the matrix represents the presence or absence of the j^{th} domain (or motif) in protein i. Note in this thesis, we only consider protein domains and not motifs because the former are easier to define and are readily available in online databases.

Gene Expression: Gene expression studies measure the expression levels (which can be thought of as *activity* levels) of all genes (i.e. the gene complement) in a given organism under varying conditions. The output of these studies can be organized in a matrix $G_{n \times d}$ where n is the number of genes and d is the number of conditions: the entry g_{ij} is the expression level of gene i in condition j. Comparison of gene expression levels under two different conditions reveals indirect information about functional roles of genes: for instance, comparisons of gene expression levels in healthy and diseased cells can be used to hypothesize which genes are involved in the given disease. Gene expression datasets are available for a large number of organisms and study conditions. GEO [24] is one of the largest repositories of publicly available gene expression datasets—as of 2010, more than 7,000 gene expression datasets are available for download.

Protein Interaction Networks: Large-scale protein interaction networks are obtained from high-throughput methods that can investigate the presence of physical interactions between a large fraction of all protein pairs in a given genome. The output of these studies can be represented by an $n \times n$ matrix A where the $(i, j)^{\text{th}}$ element, a_{ij}, represents the presence or absence of a physical interaction between proteins i and j. Some studies also report a confidence score or negative logarithm of p-value that is associated with each interaction. Physical interactions between proteins are crucial for most cellular functions. For instance, protein complexes are formed from the interaction of several proteins to carry out specific functions. Co-complex networks are similar to protein interaction networks with the difference that all proteins that are subunits of a protein complex are inferred to have a physical interaction. The BioGRID database [91] is one of the largest repositories of protein interaction data—as of 2010, it contains 251,366 unique interactions reported by over 10,000 publications.

Genetic Interaction Networks: Genetic interaction studies investigate the effect of a double mutation of a pair of genes compared to single mutations of each. For instance, synthetic lethality (which is a type of negative genetic interaction) refers to a scenario where the double mutation results in cell death whereas each single mutation result in a viable cell. Genetic interaction data is available for most of the genes in *Saccharomyces cerevisiae* [19]. This data can be represented using an $n \times n$ matrix where the $(i, j)^{\text{th}}$ entry represents the presence or absence of a genetic interaction (or the corresponding confidence score) between genes i and j.

Protein Localization: Protein localization data reveals the cellular location of proteins in a cell. The location of the protein in a cell provides specific clues about its functional role: for example, proteins that are involved in transcription need to be in the nucleus at some point in time. Localization data is represented by an $n \times d$ matrix where n is the number of genes and d is the number of subcellular compartments; each entry in this matrix represents the presence or absence of a protein in a given compartment.

2.3 Why Represent Genome-Wide Data as Networks?

The methods that we present in this thesis rely on representing genomics and proteomics datasets as networks. There are several reasons for doing so. First, networks allow for a common representation of the heterogeneous data sources, allowing their eventual integration. Second, predicting gene function with guilt-by-association essentially relies on *similarities* between genes—edges in functional linkage networks naturally capture strength of similarity between gene pairs. The dimensionality of a functional linkage network is fixed, where the number of nodes in each network equals the number of genes in a given genome, whereas, the dimensionality of a feature-based representation depends on the number of features measured in a given dataset. As well, many graph-theoretic and machine learning methods are readily adaptable to such networks.

2.4 Constructing Functional Linkage Networks

To predict gene function, we assume that we are provided with networks whose nodes correspond to genes and whose edges represent the strength of the evidence for co-functionality between the connected genes. Such networks are referred to as functional linkage networks (FLNs) or functional association networks. Here we detail the means by which genomics and proteomics datasets, such as those summarized in the previous section, can be represented as networks. Table 2.2 summarizes various types of genome-wide datasets and the corresponding functional linkage networks.

We broadly classify networks into those that are derived from interaction-based data, and those that are derived from profile-based data. The former, which include protein and genetic interactions, are naturally represented as networks. For profile-based datas, such as gene expression and protein localization, the edges in the corresponding networks are typically constructed from pairwise similarity scores. In this thesis, we do not use sequence data, however, below we also describe how sequence data can be converted to

Table 2.2: Common genomics and proteomics datasets and the corresponding functional linkage networks that are used for predicting gene function.

Network	Experimental Method	Description
Co-complex Network	Co-purification (e.g.[47])	An interaction is inferred between members of subunits in a purified protein complex.
Protein-Interaction Network	Two-hybrid (e.g.[31])	A direct interaction between two proteins is inferred based on activation of a reporter gene.
Genetic-Interaction Network	SGA(e.g.[93]), dSLAM (e.g.[72])	An interaction between a pair of genes is inferred when their double mutation results in a phenotype that is unexpected based on the single mutations of each of the genes alone.
Co-localization Network	GFP fusion (e.g.[40])	An interaction is inferred from co-localization of two proteins in the same compartment in the cell.
Co-expression Network	Microarray (e.g.[84]), SAGE[100]	An interaction is inferred if two genes are co-expressed over the same conditions.
Transcriptional Regulatory Network	ChIP-on-Chip (e.g.[51])	An interaction is inferred between a gene and its regulator based on the binding of a protein to genomic DNA of the corresponding target.
Co-inheritance Network		An interaction is inferred based on the similarity of phylogenetic profiles of a gene pair. The phylogenetic profile represents the presence/absence of a gene's homologues in other organisms.

FLNs.

The Pearson Correlation Coefficient (PCC) is one of the most commonly used similarity metrics for constructing FLNs from profile-based data. In particular, given two d-vectors $\mathbf{x}$ and $\mathbf{y}$ representing the profiles of two genes, the PCC is given as:

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sum_i^d (x_i - \bar{\mathbf{x}})(y_i - \bar{\mathbf{y}})}{\sqrt{(d-1)\sum_i^d (x_i - \bar{\mathbf{x}})^2 (y_i - \bar{\mathbf{y}})^2}} \tag{2.1}$$

where $\bar{\mathbf{x}}$ and $\bar{\mathbf{y}}$ are the means of vectors $\mathbf{x}$ and $\mathbf{y}$, respectively. In other words, the PCC is the dot product between *standardized* versions of $\mathbf{x}$ and $\mathbf{y}$ that have mean zero and standard deviation one.

Networks constructed using PCC (or other similarity metrics) tend to be dense with many non-zero, yet insignificant, edges. Since only a small fraction of gene pairs are expected to have non-zero functional interaction, it is standard practice to sparsify functional linkage networks that are derived from profile-based data [105]. A common sparsification method is to only consider the top k interactions for each gene and set the rest to zero. Under experimentation, we have adopted PCC for constructing FLN, and we sparsify the constructed networks using $k = 100$ top neighbors. In practice, such network sparsification does not degrade the performance in predicting gene function [60]. The results that appear in future sections are based on this construction (see Appendix A for details).

The methods that we describe in this thesis can also be used on networks derived from sequence similarities. A simple way to construct a network based on sequence similarities is to use pairwise sequence alignment scores (or the corresponding negative log of p-values) computed by the "BLAST" algorithm [3]. There are several other similarity metrics for assessing pairwise sequence similarities, most of which consider the number of shared substring of varying lengths (referred to as "k-mers") (e.g. [52]). More sophisticated approaches, such as the "mismatch kernel" also allow for m mismatches when considering substrings of length larger than m ([53]).

2.5 Network Terminology and Definitions

In this section, we briefly review basic network terminology and definitions used in this thesis.

Representation Network data is represented using a *graph* $G = (V, E)$ where V is the vertex (node) set, $n = |V|$ denotes the number of vertices (nodes), and E is the edge set. Its common to represent G using a weighted matrix W where the $(i, j)^{\text{th}}$ element $w_{ij} > 0$ if there is an edge between node i and j. The value of w_{ij} is the edge weight (strength of the connection) between i and j. Here we assume we have symmetric $W^{\mathsf{T}} = W$, weighted networks. An *adjacency matrix* A is a binarized version of W where the edge set is $\{0, 1\}$: that is, $a_{ij} = 1$ whenever $w_{ij} > 0$, otherwise $a_{ij} = 0$.

Node Degree The degree of node i refers to the number of neighbors of node i in W: $\sum_{j=1}^{n} I[w_{ij} > 0] = \sum_{j=1}^{n} a_{ij}$ where I is the indicator function. Weighted node degree takes into account the edge weights: $\sum_{i=1}^{n} w_{ij}$. We denote node degree by k_i and weighted node degree by d_i.

Walks and Paths A *walk* is defined as a sequence of nodes where each node is adjacent to (directly connected to) the node that precedes it and the node that follows it in the sequence. A walk that contain more than one copy of a node has a loop. The length of a walk is the number of nodes in the sequence. A *path* is a walk where no node is repeated (i.e. a walk with no loops).

Diameter The *diameter* of a network is the greatest (i.e. maximum) shortest path length (i.e. distance) between any pair of nodes.

Connectedness A network is connected if there is a path between any pair of nodes. A *connected component* is a maximally-sized sub-network of W where there is a path between any pair of nodes that are in the sub-network.

Clustering Coefficient The clustering coefficient of a node i is the proportion of neighbors of i that share an edge. If the node degree of i is k_i, then there are in total $k_i(k_i - 1)/2$ possible connections between neighbors of i. The clustering coefficient is given by $(\sum_{i \sim j \wedge i \sim k} a_{jk})/k_i(k_i - 1)$ where $i \sim j$ denotes that there is an edge between i and j in A.

2.6 Predicting Gene Function from a Network

Having defined a variety of useful data types, and a standard representation in terms of functional linkage networks, we can now investigate the task of predicting gene function from networks. In this section, we focus on existing methods for predicting gene function from a single network. In the next section, we describe methods for combining multiple networks into a single network, which can then be used to predict gene function.

Given a network $W_{n \times n}$ over all genes in a genome, and a vector of labels $\mathbf{y} = \{-1, 0, 1\}^n$ where positive genes (those known to be involved in a function of interest) are represented by $+1$, negatives are represented by -1, and unlabeled genes are represented by 0, the goal is to predict which of the unlabeled genes are likely to be positives. In gene function prediction, where negative examples are rarely available, one standard approach is to assume that all non-positive, non-test genes (or unlabeled genes), are negatives.

The output of the methods that we will describe is a vector of *discriminant* scores $\mathbf{f}_{n \times 1}$ where f_i represents the likelihood that node i is positive. If the discriminant scores are continuous, then we can classify genes by setting a threshold. The overall problem has other interesting instantiations such as predicting disease genes [2], protein-protein interactions [80], subcellular localization [70], and homologous proteins [104] from network-based data.

Previous approaches for predicting gene function from a single network can be broadly categorized as those that only consider a small neighborhood around the positive genes and those that consider the global network topology. For instance, in the *majority-vote* approach, the function of an unlabeled gene is predicted based on the function of its direct network neighbors, whereas the family of label propagation algorithms (LPA) assign continuous scores (predicted labels) to all nodes in the network that represent the frequency that *random walks* of varying length that start from a given nodes end at a positive node. Despite this, the complexity of LPA scales with the number of edges in the network and thus LPA is computationally feasible for very large networks—empirically, less than 0.1% of total possible edges are typically observed in real-world networks. As well, many studies have shown that LPA often outperforms other standard classifiers in a variety of problems [60, 109, 107, 104, 65]. Below, we review local neighborhood-based methods, several formulations of LPA, and other global based methods that have been used to predict gene function from a network.

2.6.1 Local Neighborhood-Based Approaches

Majority-vote, one of the first approaches for inference of gene function, is based on the seminal work of Marcotte and colleagues [56]. In this approach, the function of a gene is predicted based on the function of the majority of its direct neighbors. Several studies have extended this approach to include second-degree neighbors [63, 15] or consider a small neighborhood around genes [38, 108].

As a first attempt, we can calculate the discriminant score for an unlabeled gene i, denoted by f_i, as the weighted sum of the labels of its direct neighbors: $f_i = \sum_{j=1}^{n} w_{ij} y_j$. However, we can obtain better performance by first normalizing the matrix W using the weighted node degrees. Doing so results in the expression $f_i = \frac{1}{d_i} \sum_{j=1}^{n} w_{ij} y_j$ where d_i is the weighted degree of node i: $d_i = \sum_j w_{ij}$. After normalizing W, the score vector can be computed in matrix form as $\mathbf{f} = D^{-1}W\mathbf{y}$ where D is a diagonal matrix with diagonal elements d_i (i.e. $D = \text{diag}(d)$).

The matrix $P = D^{-1}W$ is known as the *Markov transition matrix* (or a *singly stochastic matrix*). Since all row sums of P equal 1, they are often interpreted as a probability distribution over *random walks* starting from a given node: for example, p_{ij} represents the probability of a random walk from node i to node j and we have $\sum_j p_{ij} = 1$. Interpreting P as the Markov transition matrix facilitates the extension of direct neighbor approach to include indirect neighbors. Furthermore, the random walk interpretation allows us to better understand local neighborhood-based methods and their relationship with label propagation.

One can easily extend the above formulation to include indirect neighbors. In particular, we can obtain the probability of a random walk of length two between all nodes in the network by computing P^2. The $(i,j)^{\text{th}}$ entry of P^2 is given by $[P^2]_{ij} = \sum_{k=1}^{n} p_{ik}p_{kj}$ and represents the probability of a random walk of length two from node i to node j. In this way, we can include P^2 when calculating the node scores as: $f_i = \sum_{j=1}^{n} p_i j y_j + \sum_{j=1}^{n} [P^2]_{ij} y_j$. Similarly, this approach can be extended to include other nodes at a distance of length r (usually $r < 4$) by noting that $[P^r]_{ij}$ represents the probability of a random walk from i to j in r steps. We note that previous approaches have shown that increasing r beyond two often results in degradation of the prediction performance (e.g. [15, 63]). We will elaborate on this point when presenting label propagation.

In the context of the above representation, several existing direct and indirect neighbor-based methods define node scores as $f_i = \sum_{j=1}^{n} p_{ij} y_j + \sum_{j=1}^{n} [\hat{P}^2]_{ij} y_j$ where $\hat{P}^2$ is obtained by an *ad-hoc* modification of P^2. For example, in the BioPIXIE [63] and other similar approaches [41, 64, 37], the second summation include walks that only go through the top m genes with highest direct neighbor scores. Another variation was presented by Chua and colleagues [15] that modified the entries in $[P^2]_{ij}$ to up-weight an edge between

i and j if there is both a direct connection and a path of length two between them.

2.6.2 Label Propagation Algorithms

In the last section, we described local neighborhood-based approaches that assign discriminant scores by merely considering direct and second-order neighbors. In this section, we describe the label propagation algorithm (LPA), a principled generalization of the local neighborhood-based approaches that consider walks of all lengths between nodes. LPA can be derived using an iterative formulation, as the solution to a specific convex optimization problem, or as the maximum a posteriori (MAP) estimation in Gaussian Markov Random Fields. Below, we will describe these three formulations of LPA.

Iterative Formulation

Intuitively, label propagation can be understood in terms of the "diffusion" of labels through the edges of the network. In its iterative formulation, labeled nodes propagate their initial labels to their neighbors and then on to neighbors of neighbors, and so on. This process can be defined using the following recursion. At iteration $r+1$, the score of node i is computed using a weighted average of the score of its neighbors at the previous iteration, and its initial label:

$$f_i^{(r+1)} = (1-\lambda)y_i + \lambda \sum_{j=1}^{n} w_{ij} f_j^{(r)} \tag{2.2}$$

where $0 < \lambda < 1$ is the model parameter and $f_i^{(0)} = y_i$ [107]. Under the condition that the eigenvalues of W are all in the range $[-1, 1]$ (we denote this condition by $\rho(W) \leq 1$)[1]

[1]The condition for the convergence is that $\rho(\lambda W) < 1$ but since $0 < \lambda < 1$ it is only required that $\rho(W) \leq 1$.

the sequence $\mathbf{f}^{(r)}$ converges to:

$$\mathbf{f} = (1-\lambda)\sum_{r=0}^{\infty}(\lambda W)^r \mathbf{y}. \tag{2.3}$$

Note that $[W^r]_{ij} > 0$ if there is a walk of length r between nodes i and j. Thus, in label propagation, discriminant scores can be interpreted as a weighted sum of walks of all lengths between the nodes. Since $0 < \lambda < 1$, the weight assigned to each walk, λ^r, decreases with increasing distance. Intuitively, LPA assigns a high discriminant score f_i to any node which is connected to the positively labeled nodes with many *short* walks. As an example, Figure 2.2 shows the scores assigned by label propagation in a *modular* network where nodes are organized into clusters. In this example, there are two positive nodes; one is a hub that connects multiple clusters and the other is a member of a single cluster. As shown, LPA assigns high scores to nodes in the top left cluster as they are connected to a positive node with walks of length both one and two.

Normalizing W ensures that $\rho(W) \leq 1$; this condition also ensures the convergence of the infinite sum above (Equation (2.3)). In particular, there are two standard matrix normalization methods: 1) symmetric normalization $S = D^{-1/2}WD^{-1/2}$ and 2) asymmetric normalization $P = D^{-1}W$ (recall that $D = \text{diag}(d)$ is the diagonal row sum matrix). These two normalizations result in two slightly different versions of label propagation. Plugging in S into (2.3), we obtain:

$$\mathbf{f} = (1-\lambda)\sum_{r=0}^{\infty}(\lambda S)^r \mathbf{y} \tag{2.4}$$

We can see that there are two major differences between LPA and the local-neighborhood method: in local-neighborhood the infinite sum is truncated at $r = 2$ and no explicit parameter is defined to model the effect of increasing walk lengths on the scores.

Given that $\rho(W) \leq 1$, we can use the following Taylor expansion $(I - A)^{-1} = \sum_{r=0}^{\infty} A^r$

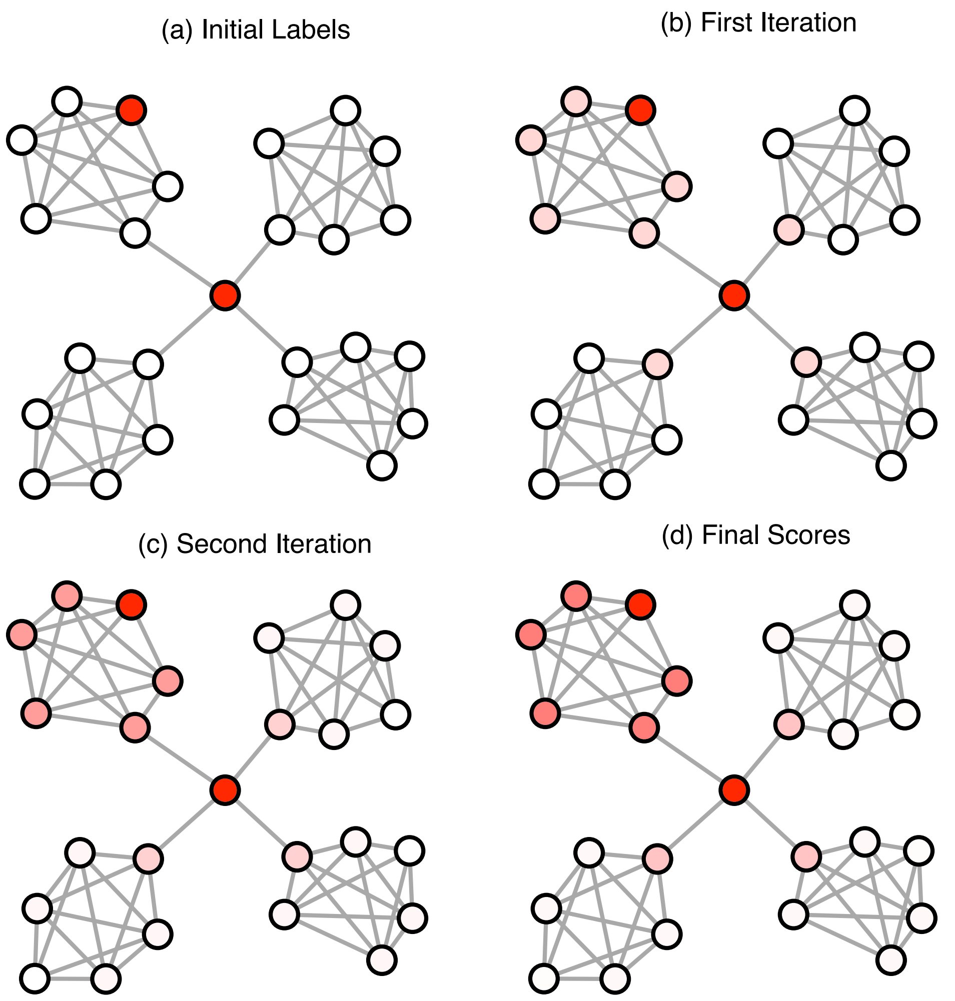


Figure 2.2: A graphical example of label propagation using two positive nodes on a modular network. Colors indicate the scores: red depicts high and white depicts low. (a) The initial labels of nodes; two positive nodes are depicted in red, and unlabeled nodes are white. (b) The scores assigned after the first recursion in Equation (2.2): $\mathbf{f}^{(1)} = \lambda W\mathbf{y} + (1-\lambda)\mathbf{y}$. (c) The scores assigned after the second recursion: $\mathbf{f}^{(2)} = \lambda W\mathbf{f}^{(1)} + (1-\lambda)\mathbf{y}$. (d) The final discriminant scores given by the exact solution to label propagation. Note that in the second recursion $\mathbf{f}^{(2)}$ all nodes reachable with a walk of length two are assigned a non-zero value, however, for those nodes not in the top left cluster this score is very small (and thus the node colors are a very faint pink).

[77], to compute the exact solution to LPA:

$$\mathbf{f} = (1-\lambda)(I-\lambda S)^{-1}\mathbf{y}. \tag{2.5}$$

Alternatively, when using P we get $\mathbf{f} = (1-\lambda)(I-\lambda P)^{-1}\mathbf{y}$. Matrix inversion has a computational complexity of $O(n^3)$. However, as we will show in Chapter 3, we can solve for $\mathbf{f}$ by using the conjugate gradient algorithm, which scales with the number of non-zero entries in the matrix S (or P).

Nabieva and colleagues [65] also proposed a variation of label propagation called *FunctionalFlow*. Their approach does not explicitly set a decay parameter λ or down-weight the influence of hubs by normalization: these criterion are implicitly enforced by always propagating to shortest-distance neighbors first and subtracting *out-flow* from *in-flow*. Unlike LPA, FunctionalFlow does not have a closed-form solution and the number of iterations is set by the user.

Convex Optimization Formulation

LPA can also be formulated as a convex optimization problem [107, 109]. LPA's objective function minimizes the squared loss between the discriminant scores $\mathbf{f}$ and the label vector $\mathbf{y}$ while ensuring that $\mathbf{f}$ is smooth on the *Laplacian* L:

$$\mathbf{f}^* = \underset{\mathbf{f}}{\operatorname{argmin}} \; (\mathbf{f}-\mathbf{y})^\mathsf{T}(\mathbf{f}-\mathbf{y}) + \lambda \mathbf{f}^\mathsf{T} L \mathbf{f} \tag{2.6}$$

where $L = (D - W)$ and D is the diagonal matrix of row sums of W. By noting that $(\mathbf{f}-\mathbf{y})^\mathsf{T}(\mathbf{f}-\mathbf{y}) = \sum_i (f_i - y_i)^2$ and $\mathbf{f}^\mathsf{T} L \mathbf{f} = \sum_{ij} W_{ij}(f_i - f_j)^2$, we can see that the first term in the LPA's objective function encourages the discriminant scores of labeled genes to be similar to their initial labels and the second term encourages discriminant scores of neighboring nodes to be similar to each other. Instead of L we can also use the normalized Laplacian: $\tilde{L} = D^{-1/2} L D^{-1/2} = I - S$ [107] which often performs better in practice (as we will show in Chapter 5). Using $\tilde{L}$, differentiating (2.6) with respect to $\mathbf{f}$ and setting the derivative to zero we obtain:

$$\mathbf{f}^* = (1-\lambda')(I-\lambda' S)^{-1}\mathbf{y} \tag{2.7}$$

where $\lambda' = \frac{\lambda}{1+\lambda}$. Using L (instead of $\tilde{L}$) results in $\mathbf{f}^* = (I + \lambda D - \lambda W)^{-1}\mathbf{y}$.

One advantage of characterizing LPA in terms of a convex optimization problem consisting of a loss function (squared error) and a regularizer (smoothness on the Laplacian) facilitates the design of its extensions. For instance, in Chapter 4 we will show how to extend this formulation to include ontology structure.

Gaussian Markov Random Field

LPA can also be written as the solution to MAP estimation in Gaussian Markov Random Fields [83]. In this formulation, the nodes in the graph represent the hidden random variables f_i's and have a joint Gaussian prior: $\mathbf{f} \sim N(0, (\lambda L)^{-1})$ (where $N(0, A)$ denotes the Gaussian distribution with zero mean and covariance A). Given an observation value y_i for each node i, the goal is to find a setting of the random variables $\mathbf{f}$ that maximizes the probability of the observations. In particular, in this context, we can find $\mathbf{f}$ by solving the following problem:

$$\begin{aligned}
\mathbf{f}^* &= \underset{\mathbf{f}}{\operatorname{argmax}}\ p(\mathbf{y}|\mathbf{f})p(\mathbf{f}) \qquad (2.8)\\
&= \underset{\mathbf{f}}{\operatorname{argmax}}\ N(\mathbf{y}|\mathbf{f}, I)N(\mathbf{f}|0, (\lambda L)^{-1})\\
&= \underset{\mathbf{f}}{\operatorname{argmax}}\ N((I+\lambda L)^{-1}\mathbf{y}, (I+\lambda L)^{-1})\\
&= (I+\lambda L)^{-1}\mathbf{y}
\end{aligned}$$

where the last equality is derived by noting that the expected value of the Gaussian distribution is also its mode. This representation suggests that the label bias y_i can be viewed as a noisy estimate of the soft label f_i. Consequently, y_i can be used to represent our prior belief about the label of node i: we will use this interpretation in Chapter 3 and 4 to set the initial labels of the unlabeled genes. Further, this representation allows us to understand the role of $M = (I+\lambda L)^{-1}$ in terms of the posterior covariance matrix; the i^{th} diagonal element of M, denoted by m_{ii}, represents the posterior *uncertainty* (variance) in the estimate f_i.

2.6.3 Other Approaches

In this section, we describe two other existing approaches for predicting gene function from a network: Discrete Markov Random Fields (DMRFs) and Support Vector Machines (SVMs). SVMs, among the most widely used supervised approaches to prediction problems, can be easily applied in this setting by converting functional linkage networks to *kernels*. As we will show, the formulations of these two methods are similar to that of LPA. However, to date, these two approaches have not performed as well as LPA or certain local neighborhood-based methods in predicting gene function [65, 89, 55, 76, 61].

Discrete Markov Random Fields

Discrete Markov Random Fields are another class of methods that have previously been used to predict binary gene function from a network [44, 23, 99, 65]. In its simplest form, the binary node *states* $\mathbf{s} = \{-1, 1\}^n$, which are used to classify genes, are computed as the MAP estimate in an Ising model:

$$\mathbf{s}^* \;=\; -\operatorname*{argmin}_{\mathbf{s}} \sum_{i,j} w_{ij} s_i s_j \;:\; \mathbf{s} = \{-1, 1\}^n \tag{2.9}$$

where the states of the positive and negative genes are constrained to be 1 and -1, respectively. In this way, the above objective function penalizes inconsistent state assignments to neighboring genes. For the binary case, Equation (2.9) can be solved efficiently using the graph cut algorithm (as done in [65]). LPA can be seen as a relaxation of Equation (2.9) where $\mathbf{s} = [-1, 1]^n$. In practice, by assigning continuous discriminant scores, LPA performs better than models based on discrete Markov random fields that restrict the node labels to be binary [65, 89, 61].

Support Vector Machines

A functional linkage network can be easily represented as a *kernel* matrix and used as input to a Support Vector Machine (SVM) (for example, as done in [11, 74]). Below, we first describe the *primal* SVM formulation and then derive the corresponding *dual*. Although the primal formulation assumes that each training example is represented by a feature vector $\mathbf{x}_i$, the dual formulation directly operates on a similarity (affinity) network K (the kernel matrix) with k_{ij} representing the similarity of examples i and j.

An SVM constructs a separating hyperplane in high-dimensions that separates the positive and negative examples. The separating hyperplane is chosen as the one that has the largest distance (*margin*) to the positive and negative examples. In particular, we can solve for the normal $\mathbf{v}$ to this hyperplane by optimizing the following problem:

$$\underset{\mathbf{v},b}{\operatorname{argmin}}\ \frac{1}{2}||\mathbf{v}^\mathsf{T}||_2^2\ :\ y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b) \geq 1,\ i = \{1, ..., n\} \tag{2.10}$$

where $\mathbf{v}$ is the vector orthogonal to the separating hyperplane (the normal to the hyperplane), b is a scalar representing its intercept, and $\mathbf{x}_i$ is the feature vector for example i. The value of $y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b)$ is the margin of the example i. Once we obtain $\mathbf{v}$, we can classify test examples j by computing $f_j = \mathbf{v}^\mathsf{T}\mathbf{x}_j + b$. Equation (2.10) requires that all positive examples fall on one side of the hyperplane and all negative examples on the other side (as required by the constraint: $y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b) \geq 1$). In the soft-margin formulation, each training example i is allowed to fall on the "wrong" side of the hyperplane subject to a penalty, measured by ξ_i. In this case, we can solve for the normal vector that defines the separating hyperplane as follows:

$$\underset{\mathbf{v},b,\xi}{\operatorname{argmin}} \ \frac{1}{2}||\mathbf{v}^\mathsf{T}||_2^2 + C\sum_i^n \xi_i \ : \ \xi_i \geq 0, \ y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i, \ i = \{1, ..., n\} \tag{2.11}$$

where C is the regularization parameter that trades off error against margin. Note that the feature vectors, $\mathbf{x}_i$'s, can be very high-dimensional; however, the dual of the above objective function allows us to only consider the dot product between feature vectors. In particular, the Lagrange function of (2.11) is given by:

$$L(\mathbf{v}, b, \xi, \mathbf{a}, \eta) = \frac{1}{2}||\mathbf{v}||_2^2 + C\sum_i^n \xi_i + \sum_i^n a_i(1 - \xi_i - y_i(\mathbf{x}_i\mathbf{v} + b)) - \sum_i^n \eta_i\xi_i \tag{2.12}$$

where $\eta \geq 0$ and $\mathbf{a} \geq 0$ are the dual variables. Differentiating the Lagrange function with respect to $b, \mathbf{v}, \xi$, setting the derivatives to zero, and substituting the appropriate variables into (2.11), we obtain the dual optimization problem:

$$\underset{\mathbf{a}}{\operatorname{argmin}} \ \frac{1}{2}\mathbf{a}^\mathsf{T}\tilde{K}\mathbf{a} - \mathbf{a}^\mathsf{T}\mathbf{1} \ : \ \mathbf{a}^\mathsf{T}\mathbf{y} = 0, \ 0 \leq \mathbf{a} \leq C \tag{2.13}$$

where $\mathbf{1}$ is a vector of ones, $\tilde{k}_{ij} = y_j k_{ij} y_i$ (or equivalently $\tilde{K} = \text{diag}(\mathbf{y})K\text{diag}(\mathbf{y})$) and $k_{ij} = \mathbf{x}_i{}^\mathsf{T}\mathbf{x}_j$. The kernel matrix K represents pairwise similarities (e.g. dot products) between the n examples. In the dual form, we can use any matrix K with $k_{ij} = \Phi(\mathbf{x}_i)^\mathsf{T}\Phi(\mathbf{x}_j)$ for some high-dimensional feature map Φ. The problem (2.13) is a quadratic optimization problem and can be solved efficiently using, for example, the the SMO algorithm [79]. See [85] for a review of SVMs and their solution methods. Once we obtain $\mathbf{a}$, we can compute the discriminant scores for a test example j as follows:

$$f_j = \sum_j a_j k_{ij} y_j. \tag{2.14}$$

A functional linkage network can be used as a kernel in the SVM formulation as long it is positive semi-definite. A network constructed by a dot product to measure pairwise similarities is symmetric and positive semi-definite and thus is a kernel. Otherwise, the *diffusion kernel* introduced by Kondor and Lafferty [46] can be used to convert any symmetric affinity matrix into a kernel. In particular, the diffusion kernel of an affinity matrix W is defined by:

$$e^{\beta L} = \lim_{n \to \infty} \left(I + \frac{\beta L}{n} \right)^n = I + \beta L + \frac{\beta^2}{2!} L^2 + \frac{\beta^3}{3!} L^3 + ... \tag{2.15}$$

where β is a parameter and L is the Laplacian of W (i.e. $L = D - W$). The construction of the diffusion kernel is based on the fact that the exponential of a symmetric matrix (i.e. e^L for a matrix L) is always positive semi-definite. Note that multiplying the diffusion kernel with the label vector $\mathbf{y}$ results in a vector similar to the one we obtain as the solution to the label propagation algorithm (see Equation (2.3)), the difference being the factorial terms in Equation (2.15). However, in the SVM formulation we pre-multiply each y_i with a_i. In practice, $\mathbf{a}$ tends to be a sparse vector with many zero entries[2] and so in the SVM formulation only a select few labeled examples "propagate" their initial labels. We note that LPA and local neighborhood-based method often result in better accuracy than SVM in predicting gene function [55, 97, 76]. One explanation for this observation is that, in this setting, we often only have a few positive examples, which is in practice insufficient for training SVMs with good generalization performance.

2.7 Predicting Gene Function from Multiple Networks

In the previous section, we have focused on methods that use a single functional linkage network to classify genes. In this section, we consider existing methods for combining d

[2]By construction, $a_i > 0$ only if $y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b) = 1 - \xi_i$. In the separable case (i.e. when $\xi_i = 0, \forall i$), $a_i > 0$ only for the *support* example (those with $y_i(\mathbf{v}^\mathsf{T}\mathbf{x}_i + b) = 1$).

functional linkage networks $\{W_1, ..., W_d\}$, each constructed from a different genome-wide dataset, when predicting gene function.

As a first attempt, we can construct a combined network by averaging the edges in the d networks (*uniform weights*) [74], $W^* = \frac{1}{d}\sum_d W_d$, and use the combined network with any of the methods presented in the previous section to obtain the discriminant scores $\mathbf{f}$. Surprisingly, the performance of this simple approach is often competitive with that of the more sophisticated approaches described below [54]. However, this method cannot detect noisy networks or account for redundancy, and in such scenarios results in degradation of accuracy [54, 60]. Further, different networks may be more suited for predicting different functions; for instance, shared domain networks are more informative of molecular functions and not so informative of biological processes. However, this uniform weights formulation assumes that all networks are equally predictive of all functions.

In this section, we describe two categories of approaches that attempt to use knowledge about gene functions when constructing a combined network. Approaches in the first category, *weighted network combination*, solve a challenging optimization task to optimize a weighted combination of networks for a given prediction task. Approaches in the second category construct a *probabilistic functional linkage network* where the edges in the networks are directly interpreted as probability of co-functionality of the connected nodes and are used as input to a local neighborhood-based method to predict gene function. The former approaches result in the state-of-the-art performance, however, their solution requires long computation times even for small genomes (e.g. eight hours for predicting gene function from five yeast networks [48]). The latter approaches are limited by the assumptions that they apply in order to construct a combined network and to make predictions from the combined network.

2.7.1 Weighted Network Combination

In this approach, the combined network is constructed as a weighted sum of the individual networks $W^* = \sum_d \alpha_d W_d$ where each network weight α_d reflects the usefulness of a network for a given prediction task. The state-of-the-art methods assign the network weights by optimizing the network combination for classification with SVM [48] or label propagation [97].

For example, Lanckriet and colleagues [48] constructs multiple kernels from various genomics data sources, and jointly optimizes the kernel weights and the performance of an SVM classifier that uses the combined kernel as input. To ensure that the combined kernel remains positive semidefinite, they constrain the network weights to be positive, $\alpha \geq 0$. Further, by requiring that $\sum_d \alpha_d = 1$ the combined network must be constructed as a convex combination of the d networks. This approach is referred to as Multiple Kernel Learning (MKL). Although MKL has been shown to result in good predictive performance, computing the solution using the semi-definite programming (SDP) approach of [48] can take hours for small genomes (e.g. yeast). An active area of research in MKL is to improve the efficiency of the seminal work of [48, 79]. More recent studies [90, 81] have proposed more efficient approaches where the MKL solution is computed by alternating between optimizing the kernel weights and training the SVM classifier that uses the corresponding combined kernel. However, these approaches still require solving for the solution of a SVM multiple times.

Alternatively, the TSS algorithm [97] solves a joint optimization problem for determining the network weights and discriminant scores using the label propagation algorithm:

$$\begin{aligned} \underset{\mathbf{f},\alpha,\gamma}{\operatorname{argmin}} \quad & (\mathbf{f}-\mathbf{y})^{\mathsf{T}}(\mathbf{f}-\mathbf{y}) + \lambda\gamma, \\ \text{s.t.} \quad & \sum_d \alpha_d = 1 \\ & \alpha_d \geq 0\ ,\ \alpha_d \mathbf{f}^{\mathsf{T}}(L_d)\mathbf{f} \leq \gamma\ ,\ \forall d \end{aligned} \tag{2.16}$$

where $\mathbf{f}$ is the vector of discriminant scores, $\mathbf{y}$ is the initial label vector, λ is a model parameter and L_d is the Laplacian for the corresponding network W_d. When there is only one network, then (2.16) is equivalent to the solution of the convex optimization formulation of LPA (Equation (2.6)); the TSS algorithm extends (2.6) to combine multiple networks. We can obtain the solution to (2.16) iteratively where each iteration requires solving a system of linear equations. As reported by the authors [97], in practice, the TSS algorithm often chooses a very sparse network combination, resulting in sub-optimal performance compared to an unweighted sum of the networks: note that $\sum_d \alpha_d = 1$ is an ℓ_1-norm constraint (since $\alpha \geq 0$), which is known to result in a sparse solution. To help address this problem, the authors proposed to use a regularization where each network weight α_d is constrained to be at least c_0: $\alpha_d \geq c_0$. Note that adding such regularization is only applicable if none of the networks are noisy or irrelevant; otherwise adding a noisy network can degrade the performance of label propagation.

2.7.2 Probabilistic Functional Linkage Networks

Another approach for combining multiple networks is to first construct edge weights between genes in individual networks, to represent the probability of co-functionality according to the positive labels. Once such probabilities are computed, they can be easily integrated to construct the combined network. The way in which individual probabilities are calculated, and combined, differentiates the various approaches described in this section. Generally, once the combined network is constructed, the approaches described below use a local neighborhood-based method (see Section 2.6.1) to predict gene function from the combined network (e.g. [95, 63, 64, 101, 41]).

Troyanskaya and colleagues [95] used a simple Bayes network with fixed structure and conditionally probability tables (CPT) to infer the posterior probability of functional linkage (co-functionality) given multiple networks $p(R_{ij} = 1|W_1, ..., W_d)$, where $R_{ij} = \{0, 1\}$ is a random variable that represents functional linkage between two genes i and j.

In the simple case of conditional independence between all networks (Naive Bayes) (e.g. as in [64]), this posterior is proportional to

$$p(R_{ij}=1|W_1,...,W_d) \propto \prod_d p(w_{ij}^{(d)}|R_{ij}=1) \tag{2.17}$$

where $w_{ij}^{(d)}$ is the observed weight (link) between i and j in network d. In [95], most of the networks were assumed to be conditionally independent given the functional linkage between the genes (R_{ij}'s); however, a few dependencies were introduced between related networks (e.g. yeast-two-hybrid and affinity precipitation). The CPTs were determined by surveying a panel of experts in yeast molecular biology.

Similarly, in the STRING [101] system the posterior probability of functional linkage, given the observed networks, is given by a Noisy OR model:

$$p(R_{ij}=1|W_1,...,W_d) = 1-\prod_d (1-v_{ij}^{(d)}) \tag{2.18}$$

where $v_{ij}^{(d)}$ represents the observed evidence for functional linkage between the two genes in dataset d. In turn, $v_{ij}^{(d)}$ values are derived from the edges (links) in dataset d. In particular, each $v_{ij}^{(d)}$ is computed using a non-linear function of the observed value $w_{ij}^{(d)}$ in network d: $v_{ij}^{(d)} = f_d(w_{ij}^{(d)}, \theta_d)$. The parameters of this function, θ_d, are fitted to each network by using as positives a large number of known functionally related genes. Note that the Noisy OR function on its own assumes that none of the networks are noisy.

The above two approaches derive their model parameter based on a large collection of co-annotations; these approaches result in a fixed probabilistic functional linkage network that are used for predicting any given function category. Myers and Troyanskaya [64] presented a similar model to that of [95] except that they used Naive Bayes (2.17) and derived the CPTs from co-annotations in a given category of interest: in their approach, $p(w_{ij}^{(d)}|R_{ij})$ is set to the frequency of observing the edge weight w_{ij} among the co-annotated genes in network d. Note that applying this approach requires discretizing the edge weights.

The Naive Bayes model assumes that all the networks are independent of each other given the functional linkages and as such does not account for redundancy between the networks. This assumption may be problematic for combining large numbers of high-throughput datasets as a large collection of them will likely be co-expression networks, and are likely to be redundant with each other. More recently, Huttenhower and colleagues [41] proposed a simple heuristic for modifying the Naive Bayes model to account for redundancy. In particular, in Equation (2.17), $p(w_{ij}^d|R_{ij})$ was replaced by $Z^{-1}(2p(w_{ij}^d|R_{ij}) + 2^{U_d})$ where Z is the normalization constant and U_d is a proposed measure of network redundancy based on the amount of shared information with other networks. In this way, they proposed to exponentially decrease the *weight* of a network based on its estimated redundancy. However, this approach is not able to directly account for the redundancy of a given edge and only considers the overall distribution of edge weights.

In summary, the approaches presented above have several shortcomings that make them less suitable for on-demand prediction of gene function from the resulting combined network; most of the above methods either assume there are no noisy or irrelevant networks (e.g. Noisy OR function) or that the networks are conditionally independent given the functional linkages between the genes in the training set. One consequence of these assumptions is that the resulting combined network is often very dense and may contain many false-positive edges.

2.8 Incorporating Ontology Structure

In the previous section, we provided a literature survey of methods for performing binary classification using one or many networks. In this section, we describe another relevant task: incorporating the hierarchical structure of classification schemes (e.g. the Gene Ontology DAG) when making predictions. This type of auxiliary information can often improve the accuracy of binary classifiers. In fact, a previous study has shown that the mere knowledge of existing annotations of a given gene is predictive of its *future* annotations based on the co-annotation patterns observed in GO [45]. To date, most classification algorithms for predicting gene function that make use of the classification hierarchy have been built on top of binary classification schemes; efforts in this area can be divided into cascaded classification approaches and reconciliation schemes that modify the predictions of independently trained classifiers. Other approaches include standard structured output classification algorithms that extend SVM; however, these methods are difficult to employ to this classification problem due to its size (tens of thousands of genes and thousands of GO categories). Below, we summarize this related work.

2.8.1 Cascade Classification

In the cascade classification scheme, one binary classifier for each category is trained to predict whether annotation in the child category is warranted given annotation in the parent category (or categories). To train an internal node or leaf category classifier, only the genes that are annotated to the corresponding parent category are used as negative examples. This is because the cascade classifiers estimate the posterior probability of annotation in a category, given the annotation in the parent category and data (as explained below). Once trained, these classifiers can be used in a cascade from roots of the hierarchy down to the most specific annotation(s) warranted. One obvious disadvantage of this approach is that it limits the number of negative examples available to train each classifier to those that are annotated to the parent category. Below, we will describe two cascade classification approaches that build upon logistic regression.

Cascade Logistic Regression

Obozinski and colleagues [69] used cascade logistic regression to predict gene functions according to GO annotations. In logistic regression, the goal is to estimate $p(y_i|\mathbf{x}_i)$ where y_i and $\mathbf{x}_i$ represent the observed value of the binary label and the feature vector for the i^{th} example. In the case of binary labels $y_i = \{0, 1\}$, the logistic regression model is:

$$p(y_i = 1|\mathbf{x}_i) = \frac{\exp(b + \mathbf{v}^{\mathsf{T}}\mathbf{x}_i)}{1 + \exp(b + \mathbf{v}^{\mathsf{T}}\mathbf{x}_i)} \tag{2.19}$$

where $\mathbf{v}$ and b are the parameters that are fit using Maximum Likelihood Estimation (MLE) [9]. Assuming that there are c categories (here, gene functions) and $\mathbf{y}_i = [y_i^{(1)} \ldots y_i^{(c)}]^{\mathsf{T}}$, and that $y_i^{(a)}$ and $y_i^{(\pi_a)}$ represent the observed value of the label for example i in the category a and its parent category π_a, in cascade logistic regression we have:

$$p(\mathbf{y}_i|\mathbf{x}_i) = \prod_{a=1}^{c} P(y_i^{(a)}|y_i^{(\pi_a)}, \mathbf{x}_i) \tag{2.20}$$

with $p(y_i^{(a)} = 1|y_i^{(\pi_a)} = 0, \mathbf{x}_i) = 0$. Fitting a logistic regression to each category is similar to fitting an independent logistic regression except that only genes annotated to the category of interest and its corresponding parent category are used as training data. In case of multiple parents, the negative set can be taken to be the genes annotated to one or both parents. At classification time, the final probabilities are estimated using (2.20) so the probability of classification in category a is equal to the product of probabilities assigned from the root to category a.

Tree Hierarchy Logistic Regression

The cascade logistic regression method defined above solves for a parameter vector for each category (class) in the hierarchy. In contrast, Shababa and Neal [86] proposed *Corr-*

MNL, a multinomial logit regression method that enforces similarity between parameter vectors $\mathbf{v}_c$ for nearby classes in the hierarchy. Their approach is designed specifically for tree hierarchies where there are no cycles (i.e. each node has exactly one parent). Multinomial logit generalizes logistic regression by allowing more than two outcomes. Given a tree over the classes where there is an edge between every $y^{(a)}$ and $y^{(\pi_a)}$, Corr-MNL learns a feature vector ϕ_{a,π_a} for each edge in the hierarchy: the regression coefficient for each class a is given by the sum of the coefficients ϕ_{a,π_a} on the path from the root node to a. For instance for a chain $a \rightarrow b \rightarrow c$ over three classes a, b, c, the value of $\mathbf{v}_c$ is $\phi_{b,a} + \phi_{c,b}$. As it is currently defined Corr-MNL is only applicable to trees and not suitable for DAG hierarchies such as GO.

2.8.2 Reconciliation Methods

One disadvantage of training binary classifiers for each gene function (category) individually is that, in such a setting, there are no guarantees on the consistency of the obtained classifications: for instance, an example (gene) can be classified into a child category and not the corresponding parent categories. Reconciliation methods first independently train binary classifiers for each category and then *reconcile* their predictions so that the true path rule is enforced. In particular, given the prediction of c independent classifiers $o_i^{(a)}$ for $a = \{1, ..., c\}$ where $o_i^{(a)}$ is the prediction of classifier a for gene i, a reconciliation method estimates the posterior probability of annotation in category a, given the annotation in all other categories $p(y_i^{(a)}|\mathbf{o}_i)$ where $\mathbf{o}_i = [o_i^{(1)} \; ... \; o_i^{(c)}]^{\mathsf{T}}$. Below, we describe two different reconciliation methods.

Bayesian Network Reconciliation

Barutcuoglu and colleagues [8] reconciled predictions of independently trained binary SVMs by using a Bayesian network. In particular, posterior probabilities were determined by using inference in a Bayesian network that modeled the structure of the GO hierarchy:

$$p(\mathbf{y}_i|\mathbf{o}_i) = \frac{p(\mathbf{o}_i|\mathbf{y}_i)p(\mathbf{y}_i)}{Z} \tag{2.21}$$

where Z is the normalization constant. By modeling the GO hierarchy using a Bayesian network, where the edges point down from the root to the nodes representing more specific functions, and assuming that the observed $o_i^{(c)}$'s are conditionally independent given $y_i^{(c)}$, the above expression can be simplified:

$$p(\mathbf{o}_i|\mathbf{y}_i) = \prod_{a=1}^{c} p(o_i^{(a)}|y_i^{(a)}) \tag{2.22}$$

$$p(\mathbf{y}_i) = \prod_{a=1}^{c} p(y_i^{(a)}|y_i^{(\pi_a)}). \tag{2.23}$$

In their work, $p(o_i^{(a)}|y_i^{(a)})$ was set by fitting two Gaussian distributions to the $o_i^{(c)}$'s, one for the positive and one for the negative examples. Although GO is a DAG, the authors in [8] were able to use exact inference to find the most likely assignments of the hidden variables $\mathbf{y}$ given the observed $\mathbf{o}$ by restricting their training data to a small number of GO categories. The authors in [69] extended this model by using variational inference to apply Bayesian network reconciliation for all GO categories.

Isotonic Regression

Obozinski and colleagues [69] compared several other reconciliation methods to the Bayes network model. Of these, the *Isotonic Regression (IR)* method performed much better then the others. Given a set of independent predictions for a gene i in c categories $\mathbf{o}_i$, IR [7] solves the following problem:

$$\underset{y_i^{(1)},\ldots,y_i^{(c)}}{\operatorname{argmin}} \sum_{a}^{c} (o_i^{(a)} - y_i^{(a)})^2 \lambda_a \tag{2.24}$$
$$\text{s.t. } y_i^{(\pi_a)} \geq y_i^{(a)},\ \forall (a, \pi_a) \in \text{GO hierarchy}$$

where λ_a's are the parameters. In particular, IR finds the most likely *consistent* scores $\mathbf{y}_i$ for all c categories. The IR problem can be solved approximately using an algorithm proposed by Burdako and colleagues [12] called the general PAV algorithm (GPAV). Obozinski and colleagues [69] found that IR results in significantly better performance compared to several other reconciliation and cascade classification methods. However, they also concluded that in practice, most reconciliation methods often result in a degradation of accuracy of binary classifiers. For instance, they found that the solution to IR was better than those obtained by independent binary logistic regression classifiers in only a few of their evaluation categories.

2.8.3 Structured Output Methods

A third approach for incorporating ontological information is to enforce a subclass hierarchy upon the *output* of a prediction method. Structured output methods are a generalization of SVM for classifying instances to a *structured output* domain where the label vectors, $\mathbf{y}_i \in \mathcal{Y}$, belong to some output space $\mathcal{Y}$ such as a classification hierarchy [5]. In particular, structured output methods represent the primal SVM problem in a feature space described by a *joint feature map* of inputs and outputs $\Phi(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{y}_i = \{0, 1\}^c$ is a binary c-vector and c is the number of gene functions. For example, a simple joint feature map is the Kronecker product of the input and output features $\Phi(\mathbf{x}_i, \mathbf{y}_i) = \mathbf{x}_i \otimes \mathbf{y}_i$. The labeled data are classified by evaluating $\mathbf{v}^{\mathsf{T}}\Phi(\mathbf{x}_i, \mathbf{y}_i)$ where $\mathbf{v}$ is the parameter vector[3]. A new instance j (test example) is classified by solving $\mathbf{f}_j = \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} f(\mathbf{x}_j, \mathbf{y})$ where $\mathcal{Y}$ is the output space containing all possible $\mathbf{y}$'s. The optimization problem of

[3]For simplicity, we have omitted the intercept b.

solving for $\mathbf{v}$ generalizes the multi-class SVM problem [20], which is given by:

$$\begin{aligned} \underset{\mathbf{v}}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{v}||_2^2 + C\sum_i^n \xi_i \\ \text{s.t.} \quad & \forall i: \ \mathbf{v}^\mathsf{T}\Phi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} \mathbf{v}^\mathsf{T}\Phi(\mathbf{x}_i, \mathbf{y}_i) \geq 1 - \xi_i \end{aligned} \tag{2.25}$$

where $(\mathbf{v}^\mathsf{T}\Phi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y}\in\mathcal{Y}\setminus\mathbf{y}_i} \mathbf{v}^\mathsf{T}\Phi(\mathbf{x}_i, \mathbf{y}_i))$ is the margin of the instance $(\mathbf{x}_i, \mathbf{y}_i)$. Multi-class SVM aims to maximize the difference between the true label and the best runner-up for each example i, increasing the *certainty* of classification. In the structured SVM approach, the minimum margin of each example is constrained by a loss $\triangle(\mathbf{y}_i, \mathbf{y}) - \xi_i$ instead of $1 - \xi_i$. For example, $\triangle$ counts the number of incorrect labels between $\mathbf{y}_i$ and the prediction $\mathbf{y}$: $\triangle(\mathbf{y}_i, \mathbf{y}) = \sum_a I[y_i^{(a)} \neq y^{(a)}]$. A more appropriate measure for a hierarchy may incorporate the distance between the true category (e.g. gene function) and the predicted category (such as the one used in [88]). One example of a structured output SVM is given by [92]:

$$\begin{aligned} \underset{\mathbf{v},\xi\geq 0}{\text{argmin}} \quad & \frac{1}{2}||\mathbf{v}||_2^2 + C\sum_i^n \xi_i \\ \text{s.t.} \quad & \forall i, \ \forall \mathbf{y} \in \mathcal{Y} : \mathbf{v}^\mathsf{T}(\Phi(\mathbf{x}_i, \mathbf{y}_i) - \Phi(\mathbf{x}_i, \mathbf{y})) \geq \triangle(\mathbf{y}_i, \mathbf{y}) - \xi_i. \end{aligned} \tag{2.26}$$

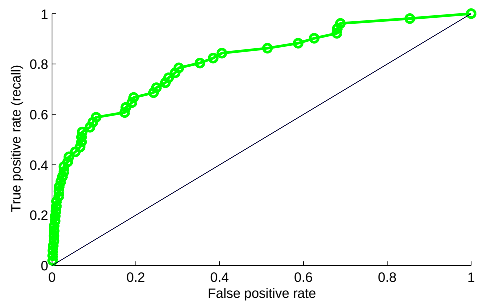
If we consider all possible output vectors $\mathbf{y} \in \mathcal{Y}$ then the number of constraints in (2.26) grows exponentially with the number of classes c. Taskar and colleagues [92] and Tsochantaridis and colleagues [96] described approximation algorithms for solving (2.26).

Sokolov and Ben-Hur [88] investigated the performance of several versions of structured output SVM in predicting gene function. Although they showed that structured output SVM improves on the performance of a binary SVM, a label propagation algorithm that solved binary classification problem out-performed both in terms of standard evaluation metrics such as precision, recall, and area under the ROC curve [88]. One explanation for this result may be the small number of training examples for many function categories in GO.

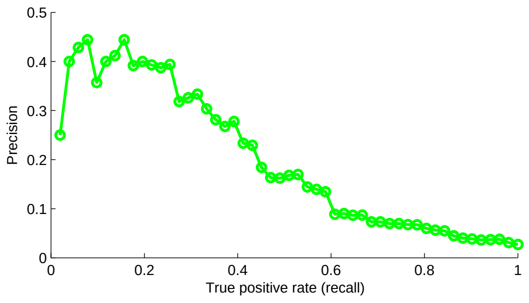
2.9 Evaluating Predictions

Consider a classifier that assigns continuous-valued discriminant scores $\mathbf{f} \in \mathcal{R}^n$; we can derive a binary classification rule from $\mathbf{f}$ by setting a threshold t and assigning all instances i such that $f_i > t$ to the positive class and the rest to the negative class. Area under the ROC curve (AUROC) and precision-recall are standard measures for evaluating the performance of a classifier that produces continuous-valued discriminant scores (or rankings) for varying settings of the threshold. Below, we will describe these metrics.

The ROC (Receiver Operator Characteristics) curve is a graphical plot of the true positive rate (TPR) as a function of false positive rate (FPR) for a binary classifier as we vary the discrimination threshold. TPR, which is also known as recall, is the number of true positives divided by the total number of positives. FPR is the number of false positives divided by the total number of negatives. The points on the curve are obtained by varying the classification threshold to retrieve an increasing number of true positives. Figure 2.3(a) shows an example ROC curve. The area under this curve (AUROC) can achieve a maximum value of 1 and a minimum of 0; a random classifier will result in AUROC of 0.5 (as shown by the black line in Figure 2.3(a)). AUROC can also be interpreted as the probability that a randomly chosen positive is assigned a discriminant score that is higher than a randomly chosen negative example. AUROC can be affine-transformed to the *Mann-Whitney* U-statistic [34]—a non-parametric statistical test for assessing whether two independent samples of observations have different medians. In particular, given n^+ true positive and n^- true negative examples (with $n = n^+ + n^-$), let f_i^+ for $i = \{1, ..., n^+\}$ denote the discriminant scores assigned to the (true) positive examples and f_j^- for $j = \{1, ..., n^-\}$ denote the discriminant scores assigned to the (true)



(a)



(b)

Figure 2.3: (a) An example ROC curve (green line); the black line shows the performance of a random classifier. (b) An example precision-recall curve.

Table 2.3: This table shows three different rankings of six examples, the associated true labels, and the average precision and AUROC assigned to each ranking.

true labels	predicted ranking	AUROC	average precision
(1,1,0,0,0,0)	(1,5,4,3,2,6)	0.63	0.7
	(2,3,1,4,5,6)	0.75	0.58

negative example. Then, the area under the ROC curve is given by [34]:

$$\text{AUROC} = \frac{1}{n^+ n^-} \sum_{i=1}^{n^+} \sum_{j=1}^{n^-} I[f_i^+ > f_j^-] \tag{2.27}$$

where $I[f_i^+ > f_j^-]$ is an indicator function for $f_i^+ > f_j^-$. The double summation counts the number of pairs (f_i^+, f_j^-) where $f_i^+ > f_j^-$. In case of tied ranks (i.e. if there exist $f_i^+ = f_j^-$), then each such instance contributes $\frac{1}{2}$ to the summation in Equation (2.27).

Precision at a given recall is defined as the fraction of predictions that are true positives and is given by TP/(TP+FP) where TP is the number of true positives and FP is the number of false positives at the given recall rate. For a classifier with continuous discriminant scores, to calculate precision at a given recall, we first set a threshold at which we retrieve the desired recall and then we compute the precision. Average precision (at varying recall levels) is equivalent to the area under the precision-recall curve where precision is plotted as a function of recall (Figure 2.3(b)).

Note that precision-recall and AUROC are not sensitive to the magnitude of the discriminant scores and only consider the rankings of the examples. Further, a classifier that performs better in terms of AUROC is not guaranteed to perform better in terms of average precision, or vice versa. In general, average precision is more sensitive to a "good" ranking of a subset of positives at low recall whereas AUROC is more sensitive to the overall ranking of all the positives. To get a better understanding of the difference between average precision and AUROC, Table 2.3 shows an example scenario where two different classifiers produced rankings for six examples with two positives. In the first ranking, one positive has the rank one (lower rank indicates higher discriminant score) and the second is ranked fifth. In the second ranking, the two positives are ranked second and third. AUROC and average precision evaluate these ranking in opposite ways; average precision prefers the first ranking and AUROC prefers the second ranking.

Chapter 3

Predicting Gene Function from Multiple Networks in Seconds

3.1 Introduction

In this chapter, we present the GeneMANIA framework for fast prediction of gene function from multiple heterogeneous data types. Our approach consists of three components: representing heterogeneous datasets as networks, combining multiple networks based on a particular prediction task, and making predictions from the combined network. In the previous chapter, we have defined a common representation of heterogeneous data types as functional linkage networks, along with existing approaches for combining such networks to make predictions about gene function. Here, we retain the functional linkage network representation and define our approach for combining multiple networks and for making predictions from the combined network. In particular, in our formulation, we decouple the network integration step from the label prediction step. However, for each step we use appropriate algorithms that make relevant assumptions. As we will show, we can combine multiple large networks (on the order of 20,000 nodes) to predict a given gene function in seconds. While fast and scalable, our formulation results in the state of the art for performance on several benchmark datasets.

The rest of this chapter is organized as follows. First we will describe several benchmark datasets that we and others have developed for evaluating the performance of gene function prediction methods. Then we describe our network integration formulation, which consists of a constrained linear regression algorithm, and propose two extensions, regularization and *simultaneous weights*, for scenarios where only a limited number of positively labeled examples are available. Next, we describe how we make predictions from the combined network using label propagation. In the results section, we first present an intuitive example of gene function prediction with GeneMANIA and then extensively evaluate the GeneMANIA algorithm on several benchmark datasets.

3.2 Benchmark Networks

To evaluate the performance of algorithms for predicting gene function from multiple networks we use several benchmark datasets for various organisms. We briefly summarize these benchmarks below—Appendix A provides a reference for each dataset.

- **MouseFunc Benchmark** consists of ten datasets and 21,603 genes including three gene expression datasets, one localization dataset, two phylogenetic profile datasets, one phenotype dataset, one protein interaction network, and two protein domain composition datasets [76]. This benchmark was used to competitively evaluate existing methods that have been developed to predict gene function.

- **Small Yeast Benchmark (Yeast15)** consists of 15 datasets and 6,413 genes including two phenotype datasets, seven gene expression datasets, one localization dataset, one transcriptional regulation dataset, two domain composition datasets, and two protein interaction networks.

- **Large Yeast Benchmark (Yeast44)** consists of 44 datasets and 3,904 genes including seven gene expression datasets, one localization dataset, and several protein interaction and genetic interaction networks. The genetic and protein interactions include all available high-throughput interactions deposited in the BioGRID database (version 2.0.45).

- **Fly Benchmark** consists of 38 networks and 13,562 genes including 32 gene expression datasets, four protein interaction networks, and two sets of domain composition data.
- ***E. Coli*** **Benchmark** consists of seven datasets and 4,175 genes constructed by [39] which include a protein interaction network, a gene expression dataset, and three datasets consisting of phylogenetic profiles and shared sequence features.
- **Human Benchmark** consists of eight datasets and 13,281 genes including two protein interaction networks, one domain composition dataset, two gene expression datasets, one phenotype dataset, and one transcriptional regulation dataset.

We construct networks from both profile-based and network-based datasets by using the Pearson Correlation Coefficient (PCC), with the exception of Yeast44 benchmark where for the network-based data we include both direct interaction networks and networks that are derived by using PCC. We set all negative PCC's to zeros—we do this for two reasons. First, negative PCC's are often not informative of co-functionality [50], and secondly, label propagation relies on having positive edge weights in the combined network. We also sparsify all networks so that each gene has at most $k = 100$ neighbors. To do so, we keep the top 100 interactions for each gene and set the rest to zero (see Appendix A for details). Subsequent to sparsification, we ensure that the networks are symmetric by setting each edge w_{ij} to the maximum of w_{ij} and w_{ji}. We sparsify the networks because the computational time of our algorithms scales with the number of non-zero edges in the networks. We have shown that sparsification does not decrease the predictive performance [60].

3.3 GeneMANIA: Fast Prediction of Gene Function from Multiple Networks

Below we first describe how we construct a combined network from multiple networks. We then describe how we use the combined network to make predictions about gene function.

3.3.1 Linear Regression for Combining Multiple Networks

Given d functional linkage networks $\{W_1, ..., W_d\}$, each over n genes, our goal is to construct a weighted combination of the networks $W^* = \sum_d \alpha_d W_d$ where the network weights $\alpha = [\alpha_1, ..., \alpha_d]$, $\alpha_d \geq 0$, are optimized for predicting a given gene function of interest. As we discussed in Chapter 2, one approach to setting the network weights, α, is to iteratively evaluate successive settings by running the overall prediction method over each resulting weighted network combination W^*. For example, we described the TSS algorithm [97] that alternates between optimizing the network combination W^* and the solution to label propagation. Although the TSS algorithm results in competitive performance and reduced computation time compared to the multiple kernel learning approach (e.g. [48]), in practice, obtaining its solution requires many iterations, where each iteration involves solving a system of linear equations with n variables. Below, we will show how to obtain the network weights, α, using constrained linear regression, instead of repeatedly consulting a prediction method as a black box. By construction, linear regression allows us to account for redundant and irrelevant networks. In our approach, obtaining α requires solving system of equations with d variables: in our setting this is advantageous because the number of networks is orders of magnitude smaller than the number of genes.

Our formulation for combining multiple networks is motivated by the *kernel target alignment* (KTA) score [21] which measures the fit of a kernel matrix (in our case affinity matrix) W for a given classification task where we must predict $\mathbf{y} = \{-1, 1\}^n$:

$$\begin{aligned}\mathrm{KTA}(W, \mathbf{y}) &= \frac{\sum_{ij} y_i y_j w_{ij}}{\sqrt{\sum_i y_i^2}\sqrt{\sum_{i,j} w_{ij}^2}} \\ &= \frac{\mathbf{y}^\mathsf{T} W \mathbf{y}}{\sqrt{n\ \mathrm{trace}(W^\mathsf{T} W)}}. \end{aligned} \tag{3.1}$$

Intuitively, KTA measures how often the edge (or similarity) between two nodes i and j, given by w_{ij}, is consistent with the labels of i and j, given by y_i and y_j. This consistency measure is then normalized by dividing by a multiple of the square root of the trace of W.

Similarly, our goal is to construct a composite network by minimizing the number of inconsistent edges. To do so, we solve the following linear regression problem:

$$\alpha^* = \underset{\alpha}{\mathrm{argmin}} \sum_{i,j=1}^{n} (y_i y_j - \sum_{k=1}^{d} \alpha_k w_{ij}^{(k)})^2,\ \alpha \geq 0 \tag{3.2}$$

Similarly to KTA, our linear regression framework penalizes each edge, w_{ij}^*'s, that connects two oppositely labeled nodes. Note that we have the constraint $\alpha \geq 0$; this is to ensure that the Laplacian constructed from W^* remains positive-semidefinite. We can write (3.2) as follows:

$$\begin{aligned}\alpha^* &= \underset{\alpha}{\mathrm{argmin}}\ \ \mathrm{trace}\left((\mathbf{y}\mathbf{y}^\mathsf{T} - \sum_d \alpha_d W_d)^\mathsf{T} (\mathbf{y}\mathbf{y}^\mathsf{T} - \sum_d \alpha_d W_d) \right) \\ &= \underset{\alpha}{\mathrm{argmin}}\ \ -2\mathbf{y}^\mathsf{T} {W^*}^\mathsf{T} \mathbf{y} + \mathrm{trace}({W^*}^\mathsf{T} W^*) \end{aligned} \tag{3.3}$$

where we used the fact that $\mathrm{trace}({W^*}^\mathsf{T}\mathbf{y}\mathbf{y}^\mathsf{T}) = \mathrm{trace}(\mathbf{y}^\mathsf{T}{W^*}^\mathsf{T}\mathbf{y}) = \mathbf{y}^\mathsf{T}{W^*}^\mathsf{T}\mathbf{y}$. We can also write the above as:

$$\alpha^* = \underset{\alpha}{\mathrm{argmin}}\ \ (\Omega\alpha - \mathbf{t})^\mathsf{T} (\Omega\alpha - \mathbf{t}),\ \alpha \geq 0 \tag{3.4}$$

where Ω is a matrix of dimensions $n^2 \times d$; each column d of Ω, denoted by Ω_d, is the *vectorized* version of network W_d; $\Omega_d = \text{vec}(W_d)$ and $\text{vec}(W)$ stacks the columns of W atop of each other; and $\mathbf{t} = \text{vec}(\mathbf{y}\mathbf{y}^\mathsf{T})$. To avoid normalizing the columns of Ω, we also estimate a bias term α_0 by including a column of $\mathbf{1}$'s in Ω. However, we discard this bias when constructing the composite network. Furthermore, in gene function prediction, there are often many more negative examples compared to positive; to address this issue we set $\hat{y}_i = \frac{-n^+}{n}$ if $y_i = -1$ and $\hat{y}_i = \frac{n^-}{n}$ if $y_i = 1$, and thus $t_i = \{\frac{(n^+)^2}{n}, \frac{-n^+n^-}{n}, \frac{(n^-)^+}{n}\}^{n^2}$.

Intuitively, as shown in Equation (3.4), our linear regression formulation treats each network W_d as a "feature vector" of dimensions $n^2 \times 1$. In this formulation, our goal is to construct a linear combination of these feature vectors that best reconstructs the target vector $\mathbf{t}$. The target vector $\mathbf{t}$ correspond to an "ideal" network where pairs of examples with the same label are connected together with a positive edge (with edge weight of $\frac{(n^-)^2}{n}$ for a pair of positive examples and $\frac{(n^+)^2}{n}$ for a pair of negative examples), and positive-negative pairs are connected together by a negative edge (with an edge weight of $-\frac{n^+n^-}{n}$). Using this intuition, our linear regression formulation constructs a linear combination of the underlying networks that result in a minimal error reconstruction of the ideal network.

We can solve for the constrained linear regression problem presented in Equation (3.4) using the active set method [49]. This method partitions the columns of Ω into two components: *active* and *inactive* sets, represented by $\Omega = [\Omega_A\ \Omega_I]$. Initially, $A = \emptyset$ and $I = \{1, ..., d\}$ where d is the number of columns in Ω. At each iteration, the indices in A and I are updated and at termination α_A is the solution to the smaller linear regression problem with Ω_A: $\alpha_A = ({\Omega_A}^\mathsf{T}\Omega_A)^{-1}{\Omega_A}^\mathsf{T}\mathbf{y}, \alpha_A \geq 0$, and $\alpha_I = \mathbf{0}$ where $\mathbf{0}$ is a vector of zeros. The termination condition is that the gradient of (3.4) is positive: $\mathbf{g} = \Omega^\mathsf{T}(\Omega\alpha - \mathbf{y}) \geq 0$. This is in fact true for $\mathbf{g}_A$; for $\mathbf{g}_I$ this condition indicates that to improve the objective, α_I must decrease (and thus become negative). In each iteration, we swap an index i from I to A which has the most negative g_i. Because d is often in the order of low hundreds we can solve for the solution very fast as we just have to solve a $d \times d$ linear system at each iteration. In practice, we use a modified version of active set where we start with $\Omega_I = \emptyset$ and $\Omega_A = \Omega$. In each iteration, we remove all indices J in A that are associated with a negative coefficient $\alpha_J \leq 0$ and add them into I. This approach is faster than the original active set method as we now remove several indices from the active set to the inactive set at once, and thus we need fewer iterations to compute the solution. Empirically, only a few iterations are required to obtain $\alpha_A \geq 0$. To ensure that our method has converged to the optimum solution, at termination we check for the condition $\mathbf{g} \geq 0$, which is always satisfied in our experiments.

Regularized Regression

As we will show in the results sections, the above linear regression framework works well when we have a sufficient number of positive examples (more than 30). However many gene functions of interest only have a few positives (three to ten). To improve the performance of linear regression in this setting, we investigate several regularization methods of the following form:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} \left(\Omega\alpha - \mathbf{t}\right)^{\mathsf{T}} \left(\Omega\alpha - \mathbf{t}\right) + J(\alpha),\ \ \alpha \geq 0 \tag{3.5}$$

where $J(\alpha)$ is the regularization term. In our experiments, we compare ridge, ridge with prior, Lasso, and elastic net regularization. Below, we will describe each of these regularization methods.

Ridge Regularization Ridge regression penalizes the ℓ_2 norm, i.e. $J(\alpha) = \lambda_1(\alpha^{\mathsf{T}}\alpha)$, where λ_1 is a scalar that determines the strength of the regularization. Ridge regression corresponds to placing a Gaussian prior with mean zero and unit variance on α; that is, $p(\alpha) \sim N(\mathbf{0}, I)$. In addition, we also investigate placing a Gaussian prior on α that has a mean $\mathbf{v}$ and diagonal precision (which is equal to inverse diagonal covariance) matrix S,

$p(\alpha) \sim N(\mathbf{v}, S^{-1})$, resulting in the following regularization term $J(\alpha) = \lambda_1(\alpha - \mathbf{v})S(\alpha - \mathbf{v})$; we will refer to this regularization as ridge with prior.

We can obtain the solution to ridge as $\alpha_A = ({\Omega_A}^{\mathsf{T}}\Omega_A + \lambda_1 I)^{-1}\Omega^{\mathsf{T}}\mathbf{y}$ and ridge with prior as $\alpha_A = ({\Omega_A}^{\mathsf{T}}\Omega_A + \lambda_1 S_A)^{-1}{\Omega_A}^{\mathsf{T}}(\mathbf{y} - \lambda_1 S_A \mathbf{v})$ for the positive components $\alpha_A > 0$. In practice, we set $\mathbf{v}$ in two different ways: *uniform-prior* and *mean-prior.* In uniform-prior, we assume a uniform prior on all d networks $v_d = \frac{1}{d}$: in mean-prior we first solve unregularized linear regression on several function (GO) categories and set v_d to the mean of network d's unregularized coefficients (mean of α_d's). We set S to a diagonal matrix with $s_{dd} = \text{trace}({W_d}^{\mathsf{T}}W_d)^{-1/2}$. We chose this setting for S so as to penalize networks with fewer non-zero edges more than those that have a larger number of non-zero edges; our assumption is that it is harder to set accurate weights for sparser networks rather than denser ones.

Lasso and Elastic Net Lasso regularization penalizes the ℓ_1 norm: $J(\alpha) = \lambda_1 \sum_d |\alpha_d|$. The Lasso regularization encourages a sparse solution where many of the coefficients α_d are set to zero. Elastic net regularization [110] combines the ℓ_1 and ℓ_2 norm penalty $J(\alpha) = \lambda_1 \sum_i |\alpha_i| + \lambda_2 \sum_i \alpha_i^2$. In [110], it was shown that elastic net results in a sparse solution and often performs better than the Lasso.

To solve the Lasso problem we use the Least Angle Regression (LARS) algorithm [25], a modified version of forward stagewise regression [35] that solves for the solution α in d steps by iteratively adding or removing a covariate from the *Active set.* LARS produces the entire solution path for Lasso (from a model with 1 to d covariates) and a simple modification of this algorithm allows us to incorporate the non-negative constraint on α [25]. Zou and Hastie [110] derived a formulation of the elastic net that can also be solved by LARS.

Simultaneous Weights for Combining Multiple Networks

As we will show in the results section, the ℓ_2 norm regularization with *mean-prior* drastically improves performance in categories with small number of annotations. However, assigning this prior requires solving several regression problems to set the mean vector $\mathbf{v}$. Here, we define a simple modification to the original linear regression problem in (3.4) that improves the performance without increasing the computational time and show that it performs better than previous approaches. In particular, in Simultaneous Weights (SW), instead of assigning network weights for each category separately, we fit the network weights to a set of related function categories. To do so, we assign the network weights by solving the following problem:

$$\alpha^* \quad = \quad \underset{\alpha}{\operatorname{argmin}} \sum_{c=1}^{h} (\Omega\alpha - \mathbf{t}_c)^\mathsf{T}(\Omega\alpha - \mathbf{t}_c), \;\; \alpha_d \geq 0$$

where $t_c = \text{vec}(\mathbf{y}_c{\mathbf{y}_c}^\mathsf{T})$, $\mathbf{y}_c$ is the label vector for function (categories) c, and $c = \{1, ..., h\}$ are h categories (or gene sets) that are related to each other. In the results sections, we show several methods for grouping related categories. Once we obtain α^*, we construct W^* and use it to predict all h categories.

We can write the above as:

$$\alpha^* = \underset{\alpha}{\operatorname{argmin}} -2\alpha^\mathsf{T}\Omega^\mathsf{T}\tilde{\mathbf{t}} + h\alpha^\mathsf{T}\Omega^\mathsf{T}\Omega\alpha$$

where $\tilde{\mathbf{t}} = \sum_{c=1}^{h} \mathbf{t}_c$ and so we only need to solve the regression problem once to get the simultaneous weights. As such, for each category, t^c_{ij}, takes on one of the three possible values: $(\frac{n_c^+}{n})^2, (\frac{n_c^-}{n})^2, -\frac{n_c^- n_c^+}{n^2}$ when i, j are both negative, both positive, and have the opposite signs, respectively, and $n_c^+(n_c^-)$ is the number of positives (negatives) in category c. As we will show in the results section, combining weights by SW improves the performance of the composite networks in predicting the relevant h gene categories while reducing the computation time (as now we are only required to solve for the network weights once when predicting h categories).

3.3.2 Predicting Gene Function from the Combined Network

To predict gene function from the combined network, we solve a binary transduction problem with the label propagation algorithm (LPA). That is, given a label vector $\mathbf{y} = \{-1, k, 1\}^n$ where positive examples are represented by $y_i = 1$, negative examples are represented by $y_i = -1$, and un-labeled genes are assigned $y_i = k$, we compute the discriminant scores $\mathbf{f}$ using LPA by solving:

$$\begin{aligned}\mathbf{f}^* &= \underset{\mathbf{f}}{\operatorname{argmin}}\ (\mathbf{f}-\mathbf{y})^\mathsf{T}(\mathbf{f}-\mathbf{y}) + \lambda \mathbf{f}^\mathsf{T} L^* \mathbf{f} \\ &= (I + \lambda L^*)^{-1}\mathbf{y}\end{aligned}$$

where L^* is the Laplacian constructed from the symmetrically normalized combined network $W^* = \sum_d \alpha_d W_d$. As we will show, using the Conjugate Gradient algorithm we can solve for the solution to LPA in seconds.

In [107, 109, 97], the initial label of un-labeled nodes is set to zero, that is $k = 0$. We set the initial labels of un-labeled genes as $k = \frac{n^+ + n^-}{n^+ + n^-}$ where n^+ and n^- are the number of positive and negative examples, respectively. This modification results in better performance with unbalanced classification problems such as gene function prediction [60]. We have investigated two different methods for setting the negative examples; we use all non-positive, non-test genes as negatives, we also investigate setting negatives as those genes annotated to a sibling class in GO.

Efficient Implementation of Label Propagation

As shown above, we can obtain the solution to the label propagation by solving a system of linear equations, $A\mathbf{f} = \mathbf{y}$ for example where $A = (I + \lambda L)$. While in principle this

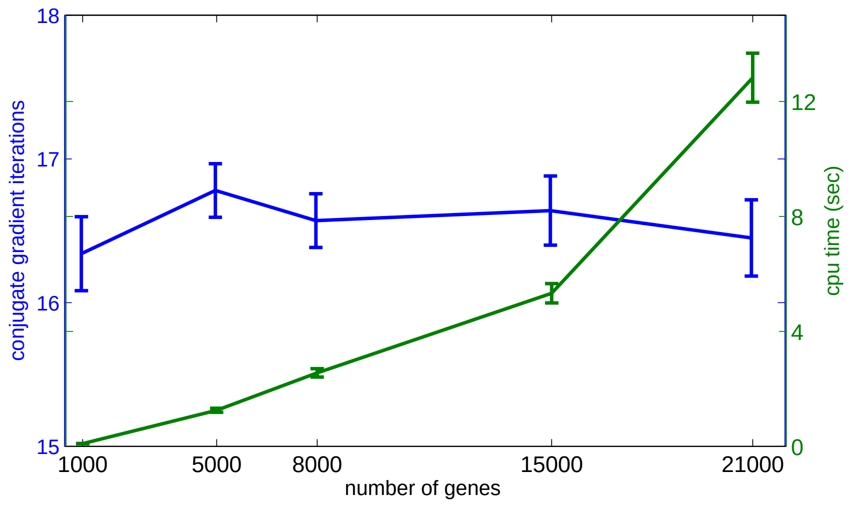


Figure 3.1: The number of CG iterations required to compute the label propagation solution as a function of the number of genes (nodes). Left axis: the number of CG iterations. Right axis: computation time of label propagation. Experiments were run using a co-expression network from the MouseFunc benchmark data. The final point on the plot used the full mouse gene complement (for which data are available), and the other gene numbers were derived by taking random subsets of the full gene complement. Distribution is over 100 randomly selected GO categories. The quadratic dependence of computation time on the number of genes is due to the quadratic growth in number of non-zero association links in the networks as a function of the number of genes.

system can be easily solved by multiplying the inverse of matrix A by $\mathbf{y}$, doing so has computational complexity in the order of $\mathcal{O}(n^3)$. However, for a sparse and symmetric matrix A, we can use the conjugate gradient (CG) algorithm with $\mathcal{O}(m)$ complexity where m is the number of non-zero entries in A. CG is an iterative algorithm, each iteration requiring a matrix-vector and vector-vector products; though it is guaranteed to converge in n steps (n being the number of nodes), in practice it converges in a few steps (e.g. 20 iterations on all experiments described below). Here, we briefly describe the CG algorithm—see [68] for details. In Chapter 5, we provide some reasons why CG converges so quickly.

Similarly to the Gradient Descent method, CG uses the fact that the solution to

$A\mathbf{f} = \mathbf{y}$ is also a unique minimizer of:

$$\underset{\mathbf{f}}{\operatorname{argmin}} \ \frac{1}{2}^{\mathsf{T}}\mathbf{f}A\mathbf{f} - \mathbf{f}^{\mathsf{T}}\mathbf{y}. \tag{3.6}$$

In each CG iteration the current estimate of the solution $\mathbf{f}^{(t)}$ is updated as follows: $\mathbf{f}^{(t)} = \mathbf{f}^{(t-1)} - \alpha_t \mathbf{p}^{(t)}$ where α_t is the search step (a scalar) and $\mathbf{p}^{(t)}$ is the search direction. In Gradient Descent, the current search direction coincides with the residual (gradient of $\mathbf{f}$ in (3.6)) given by $\mathbf{r}^{(t)} = A\mathbf{f}^{(t-1)} - \mathbf{y}$. In contrast, CG ensures that $\mathbf{p}^{(t)}$ is conjugate to $\mathbf{p}^{(j)}, j < t$, with respect to A (that is $\mathbf{p}^{(j)\mathsf{T}} A\mathbf{p}^{(t)} = 0$). This criterion results in faster convergence of CG compared to Gradient Descent [68].

In practice, we have consistently observed that CG converges in fewer than 20 iterations. Figure 3.1 shows the average number of CG iterations and computation time on a network of varying size (in terms of number of nodes). As shown, the maximum number of CG iterations observed in any test was 20 and the maximum computation time was 15 seconds.

3.4 Results

We have organized our experiments in four sections. First, we present an intuitive example of predicting gene function with GeneMANIA. Next, we report the performance of GeneMANIA, as well as eight other existing approaches, on the MouseFunc challenge; as well, we compare the performance of GeneMANIA, the TSS algorithm, and BioPIXIE on a smaller set of yeast networks (these results are from [60, 76]). We then investigate the performance of various regularization methods and SW on yeast networks, and compare SW with unregularized linear regression on mouse, human, fly, and E.coli networks (these results appear in [59]). Finally, we describe a slightly different biological application of GeneMANIA; we show how we can use GeneMANIA to quantify the “uniqueness” of functional information in a given genomic dataset, in the context of all existing datasets.

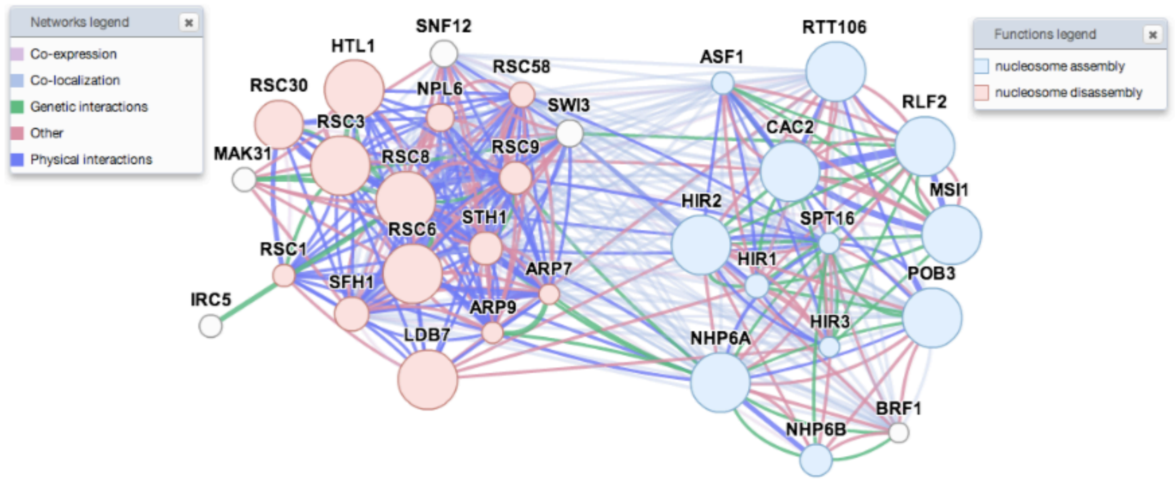


Figure 3.2: Predicting genes involved in nucleosome assembly and disassembly in yeast using GeneMANIA. The query (positive) genes consist of eleven genes represented by the largest circles (HTL1, RSC3, RSC8, RSC6, LDB7, POB3, RLF2, FTT06, CAC2, HIR2, NHP6A). The graph shows the local neighborhood around the query genes as well as the top 20 predictions. The combined network is constructed from co-expression, co-localization, genetic and protein interactions, and shared phenotype data (shown as "others" in the legend). Fifteen of the top 20 predictions are either involved in nucleosome disassembly (pink nodes) or assembly (blue nodes).

3.4.1 Predicting Nucleosome Remodeling Genes

In this section, we present a graphical example of gene function prediction and the benefits of combining multiple networks. In this example, our goal is to predict genes that are involved in *nucleosome organization (NO)*. Formally, in GO, the nucleosome organization function is defined as "a process that is carried out at the cellular level which results in the assembly, arrangement of constituent parts, or disassembly of one or more nucleosomes". In particular, nucleosome assembly and disassembly, the two main descendant categories of NO, have a great impact on gene regulation as nucleosome remodeling is a necessary step in DNA transcription, replication, and repair [10]. As of 2010, in yeast, there are 51 genes annotated with nucleosome organization, 19 with nucleosome disassembly (ND) and 18 with nucleosome assembly (NA).

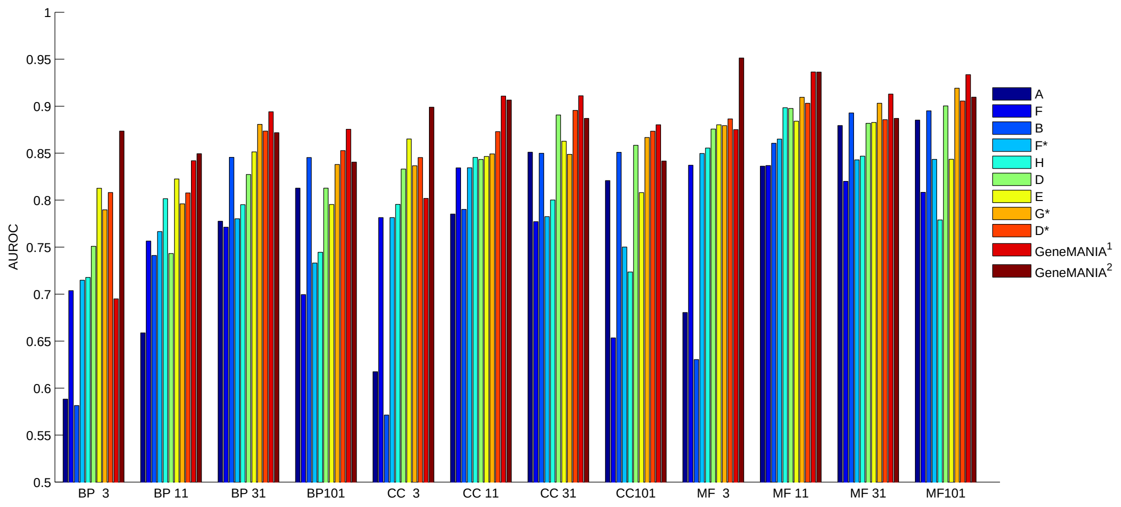
We use the GeneMANIA prediction server [103] to predict genes involved in nucleosome assembly or disassembly (a total of 39 genes), using only 11 positive genes as input (we randomly selected 6 positive genes annotated to nucleosome disassembly and 5 annotated to nucleosome assembly). Figure 3.2 shows the local network around the top 20 predicted (high-scoring) genes. Note that co-localization and co-expression networks play an important role in linking together NA and ND genes whereas ND genes are co-complexed with each other. Among the top 20 predictions, nine were involved in ND and six were involved in NA. Three of the predicted genes are annotated to chromatin organization but don't have any specific annotations and two of the predicted genes have no significant annotations related to chromatin remodeling. As shown in this figure, using multiple network types allows us to identify genes involved in biological process that may span more than one complex.

3.4.2 Evaluating GeneMANIA

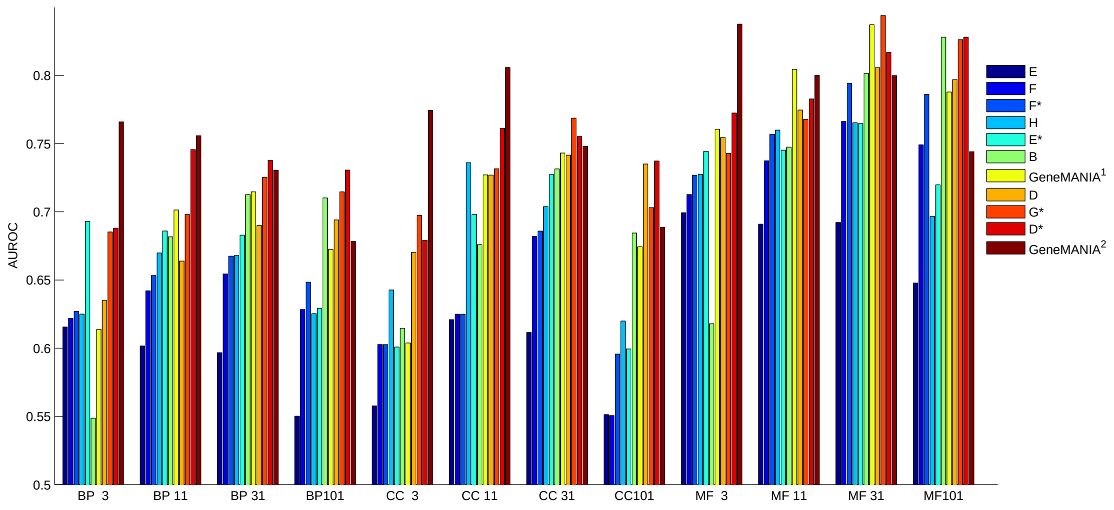
In this section, we extensively evaluate the GeneMANIA algorithm on the MouseFunc and the Yeast15 benchmarks. As we will show later, other methods that achieve similar performance as GeneMANIA require orders of magnitude longer computation times.

Performance on MouseFunc

In the MouseFunc challenge [76] the participants were provided with ten high-throughput datasets consisting of 21,063 mouse genes and function annotations for 1,726 GO Biological Process (BP), 326 GO Cellular Components (CC), and 763 GO Molecular Functions (MF). The contest consists of predicting confidence scores for all genes for each of these GO categories. The evaluation is based on 1) prediction of gene function for a random set of genes whose annotation was left out ("test set") and 2) prediction of gene function for a set of genes that had acquired new annotations in the span of a year when the training data was collected from GO ("novel set"). Each participant was allowed to make two submissions (entries) to the challenge; the second submission was due after all participants received feedback on their performance.

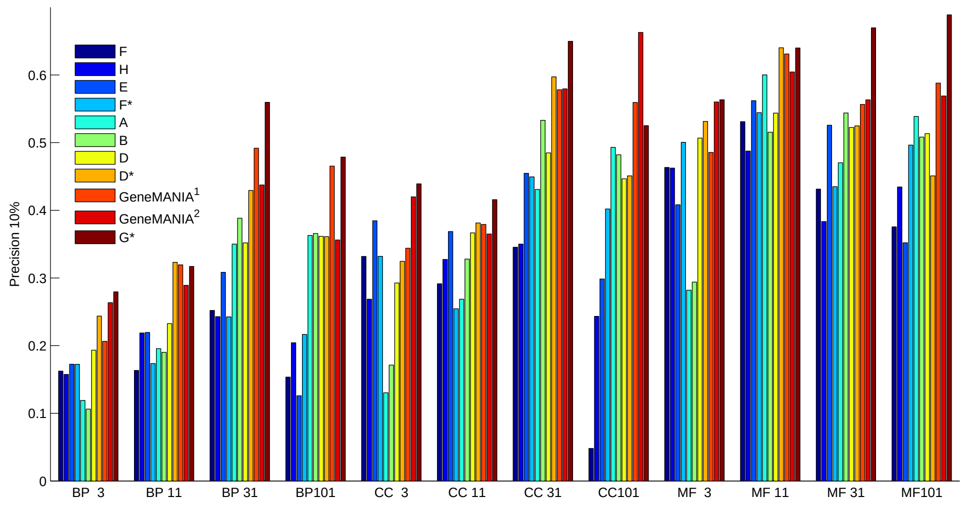


(a) Test Set

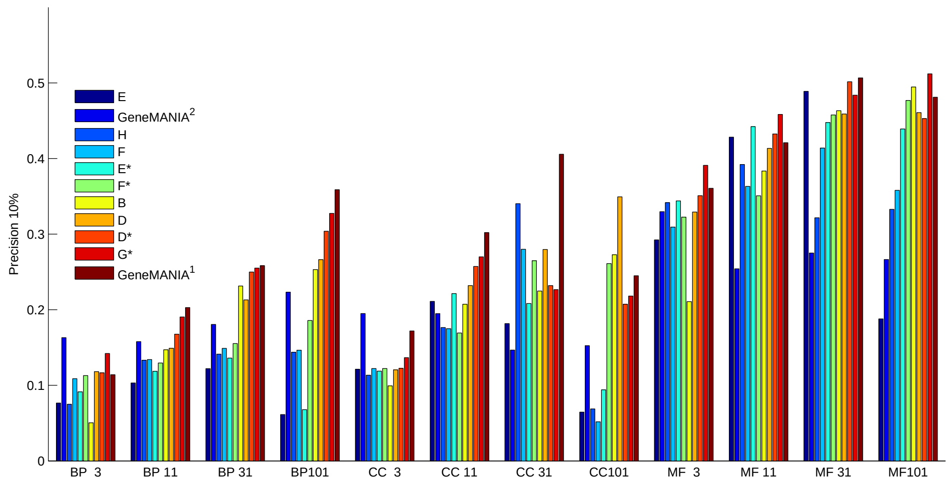


(b) Novel Set

Figure 3.3: Performance of GeneMANIA first (GeneMANIA^1) and second (GeneMANIA^2) submission to MouseFunc as well as the performance of other participants in terms of area under the ROC curve. The second submissions are labeled by adding a star $*$ to the team name. The performance is shown as the average in 12 evaluation categories. Each evaluation category is constructed by a pairwise combination of GO hierarchy (BP, CC, MF) and specificity (number of annotations [3-10], [11-30], [31-100], [101-300]); for example BP3 shows the average performance in predicting BP categories with 3-10 annotations.



(a) Test Set



(b) Novel Set

Figure 3.4: Performance of GeneMANIA first (GeneMANIA^1) and second (GeneMANIA^2) submission to MouseFunc as well as the performance of other participants in terms of precision at 10% recall. The second submissions are labeled by adding a star $*$ to the team name. The performance is shown as the average in 12 evaluation categories.

In our first submission (GeneMANIA^1) we used unregularized linear regression for combining networks and made predictions from the combined network using label propagation, with the initial label of all non-positive examples set to -1 (i.e. initially we assume all non-positives are negatives). In our second submission (GeneMANIA^2) we made two changes: 1) for each GO category c we only used genes annotated to a sibling category of c, but not to c, as a negative examples and assumed all other non-positive genes are unlabeled, 2) we used ridge regularization with mean-prior where we set the prior to average linear regression weights when predicting all categories in a given GO hierarchy (thus we have three different priors: one for predicting BP, one for predicting CC, and one for predicting MF categories).

We follow the same process as [76] and compare prediction accuracies using average area under the ROC curve (AUROC) and precision at 10% recall in twelve MouseFunc evaluation categories. Briefly, the evaluation classes are created by grouping GO categories corresponding to all pairwise combinations of the three GO branches (BP, cellular component [CC], and molecular function [MF]) and category size range based on number of annotations (categories with [3 to 10], [11 to 30], [31 to 100], and [101 to 300] annotations). We report prediction performance on both the "test" and "novel" benchmarks.

Figure 3.3 shows the performance of GeneMANIA's first and second submission as well as the performance of other participants in terms of AUROC. The bars are sorted based on average performance on all evaluation categories. On the test set, either GeneMANIA^1 or GeneMANIA^2 achieves the best performance on all evaluation categories. On the novel set, GeneMANIA achieves the best performance in 9 of the 12 evaluation categories. Note that in general GeneMANIA^2 drastically improves performance for categories with a small number of annotations (BP3, CC3, MF3). However, GeneMANIA^1 has better performance on larger categories. The reduced performance of GeneMANIA^1 in the categories with the fewest annotations is likely due to overfitting, because we used unregularized linear regression to set the network weights. In GeneMANIA^2, we switched to ridge regression with mean-prior, which is less prone to overfitting. We suspect that one reason for the drop in prediction performance of our second submission on the larger GO categories in the test benchmark is because of our definition of negative examples. In our first entry, we defined as negative examples all genes with any GO annotation but not one in the category being predicted. In our second entry, we refined this definition so that the negative examples for a given category were only those annotated to a sibling category, that is, one that shared a parent in the GO hierarchy. Although choosing the genes annotated to sibling categories of interest as negatives improved prediction performance on novel tasks, it degraded the prediction performance on the test set. This may due to the reduction in the number of negative examples. One way to alleviate this effect is to define a range of label biases between [-1, 0] for genes that have GO annotations but are not annotated to the function of interest.

Figure 3.4 shows the performance of the various methods in terms of precision at 10% recall. In terms of precision, GeneMANIA achieves the second-best performance or better in most categories. Similarly, our second submission performs better than the first when there are a small number of annotations; for larger categories our first submission often outperforms GeneMANIA[2]. As described in Chapter 2, AUROC and precision-recall prefer different rankings: AUROC prefers a ranking whereby most true positives have a “reasonable” recall whereas precision at 10% recall only considers the ranking of 10% of the positives. Comparing Figures 3.4 and 3.3, we conclude that GeneMANIA provides a better overall ranking of true positive genes whereas submission G^* has better precision. However, we note that a classifier dominates in ROC space (i.e. the ROC curve always lies above) if and only if it dominates in precision-recall space [22]; thus this result shows that there are some recall values where GeneMANIA results in better precision compared to submission G^*.

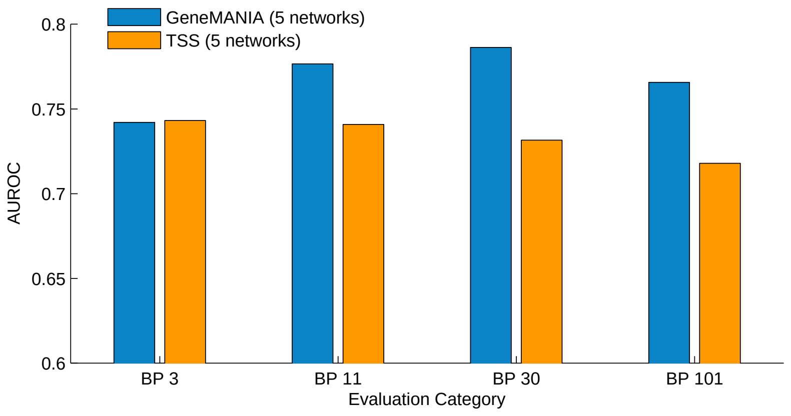


Figure 3.5: Comparison of GeneMANIA with the TSS algorithm on a yeast benchmark dataset consisting of five networks. The performance is measured using 3-fold cross-validation. These networks were used by both [48] and [97]. The bars show the average performance on 400 GO BP categories with [3-10] (BP3), [11-30] (BP11), [31-100] (BP31), and [101-300] (BP101) annotations.

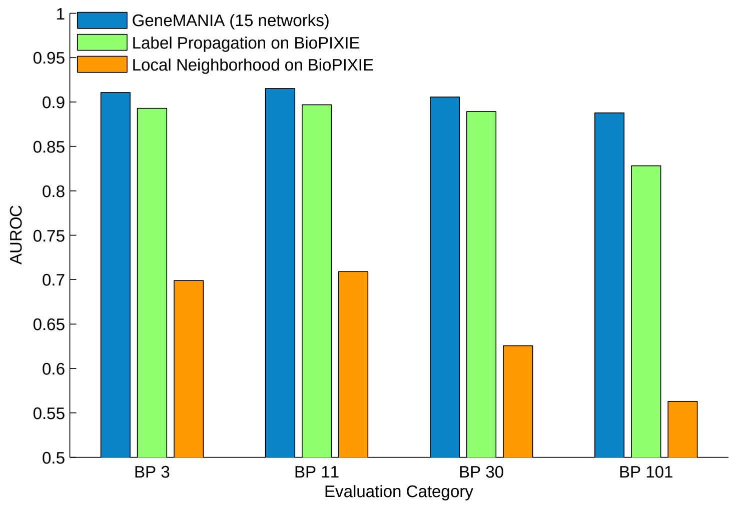


Figure 3.6: Comparison of GeneMANIA with the BioPIXIE network and local neighborhood-based search method on 400 GO BP categories with [30-10] (BP3), [11-30] (BP11), [31-100] (BP31), and [101-300] (BP101) annotations. The performance is measured using 3-fold cross-validation. GeneMANIA shows the performance of GeneMANIA on the Yeast15 benchmark. We also compared the performance of BioPIXIE local neighborhood-based method with label propagation applied to the same BioPIXIE network. The BioPIXIE network is a fixed combined network constructed from hundreds of small- and large-scale genomics and proteomics publications [63].

Performance on Yeast Networks

We also compared GeneMANIA with BioPIXIE [63] and the TSS algorithm [97]. To compare GeneMANIA with the TSS algorithm we used five yeast functional linkage networks (obtained from [97]) and created an extended evaluation scheme with 400 GO categories that we derived from a 2006 version of GO annotations (Figure 3.5). We selected a random set of 100 BP categories with [3-10], [11-30], [31-100], and [101-300] annotations. We report the performance in terms of AUROC using 3-fold cross-validation. We chose to use this smaller five network benchmark as it was also used in [97, 48] to evaluate the TSS and the MKL approaches. In particular, using this benchmark, Tsuda and colleagues [97] showed that the TSS algorithm performs comparably to the more computationally intensive MKL approach [48]. As shown in Figure 3.5, GeneMANIA considerably improves on the predictive performance of TSS as the number of positive examples increases.

We compared GeneMANIA (ridge with mean-prior) with the BioPIXIE network because it is currently deployed on a popular website that provides gene function predictions. As described in Chapter 2, the BioPIXIE algorithm [63] first combines multiple networks using a simple Bayesian network and then makes predictions from the combined network by using a local neighborhood-based method. In our experiments, we evaluate both the BioPIXIE network, which is available for download[1], and the local neighborhood-based method for making predictions from the BioPIXIE network [63]. Figure 3.6 shows the performance of label propagation (as in GeneMANIA) on the BioPIXIE network as well as the local neighborhood-based method deployed on the BioPIXIE algorithm. As shown, extracting the combined network used in BioPIXIE and then applying label propagation results in a considerable improvement over the local neighborhood-based method. We note that the BioPIXIE network was built using hundreds of low- and high-throughput datasets, so the reported performance is with regard to the specific published network, not the Bayesian network algorithm used to derive the network. In addition, to illustrate the accuracies attainable with additional datasets, we performed function prediction with GeneMANIA using a 15 yeast benchmark data (GeneMANIA (15 networks)) that we derived from recent genomics and proteomics data sources; as

[1]We obtained the BioPIXIE network from http://avis.princeton.edu/pixie/documents.php

shown, using only these 15 networks GeneMANIA can achieve similar or better performance than BioPIXIE, which is constructed from a much larger collection of data sources.

Time Requirement

GeneMANIA requires orders of magnitude less computation time as compared to existing approaches that achieve similar predictive performance. As shown in Figure 3.7, on both Mouse and Yeast15 benchmark, GeneMANIA requires less than 20 seconds of computation time. The efficiency of GeneMANIA can be attributed to both of the following factors: (a) our linear regression formulation for solving for the network weights, and (b) fast convergence of the conjugate gradient algorithm (as shown in Figure 3.1) when solving for the solution of LPA.

Figure 3.8 shows the average time requirement (in seconds) as a function of average AUROC for GeneMANIA, TSS, and BioPIXIE. Corresponding to the experiments described above, for the TSS algorithm, the performance is shown on the five network benchmark. For GeneMANIA, the figure shows the performance using the same five networks as well as the larger Yeast15 benchmark. For BioPIXIE, the figure shows the performance of BioPIXIE network when used as input to (a) local neighborhood search described in [63] and to (b) GeneMANIA's label propagation algorithm. On the same input as TSS (five network benchmark), GeneMANIA achieves better performance with less computation time. We were not able to obtain results for TSS on the 15 network benchmark as its current implementation (provided by [97]) did not converge in a reasonable amount of time. Although the BioPIXIE local neighborhood search is fast, its predictive performance is much lower than GeneMANIA's label propagation applied to the same network. Furthermore, GeneMANIA on the 15 network benchmark achieves higher accuracy and lower computation time compared to label propagation applied to the much denser BioPIXIE network.

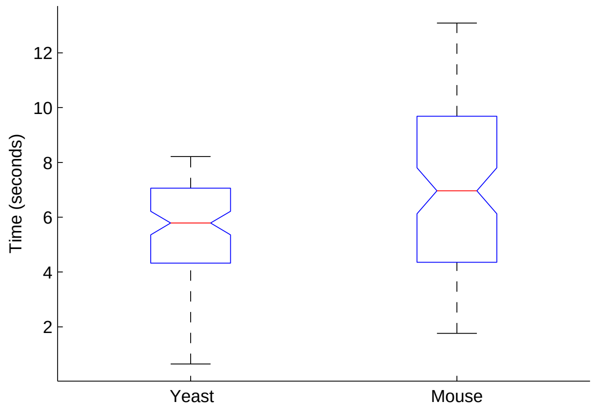


Figure 3.7: Time requirement (in seconds) of GeneMANIA to predict gene function from multiple networks. The time requirement is shown for predicting 100 random gene functions with the Yeast15 and the MouseFunc benchmark networks on a standard laptop computer (2.4GHz and 4GB memory). On each box, the central mark is the median time for predicting 100 randomly selected GO terms, the edges of the box are the 25th and 75th percentiles, and the whiskers extend to the most extreme values.

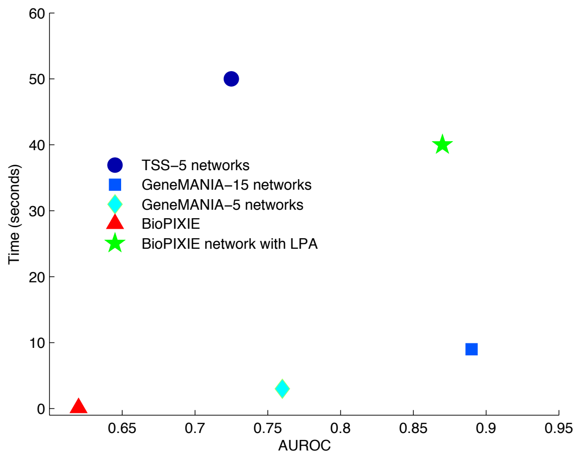


Figure 3.8: Computation time and predictive performance of GeneMANIA (with 5 and 15 networks), TSS (with 5 networks), BioPIXIE network and algorithm (BioPIXIE), and BioPIXIE network with label propagation (BioPIXIE network with LPA).

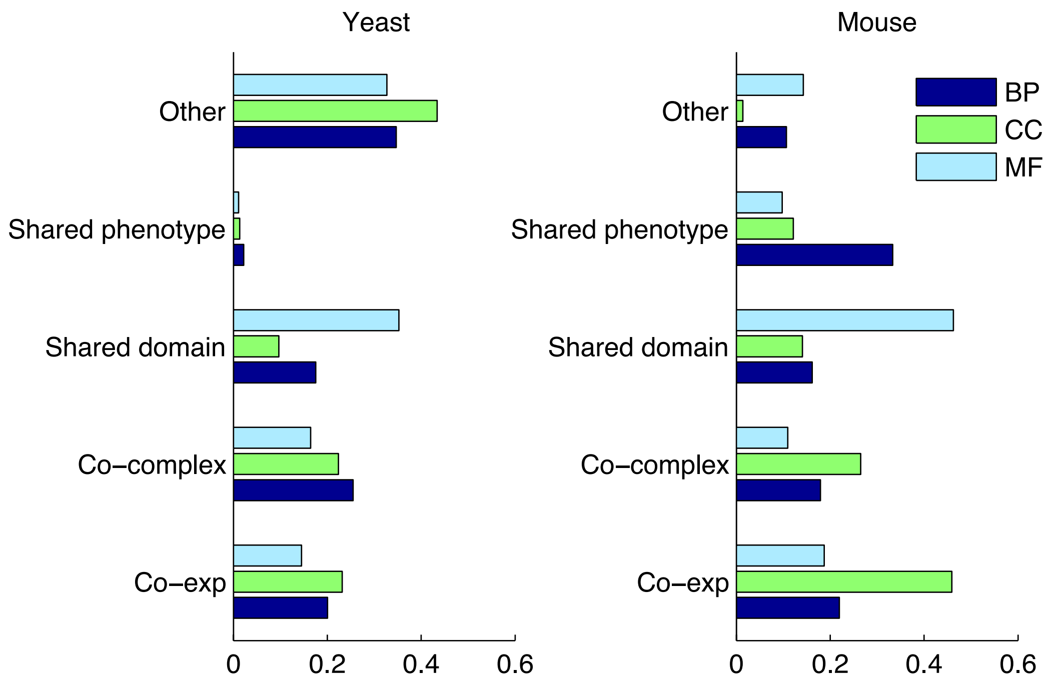


Figure 3.9: Proportion of network weights assigned to different network types by GeneMANIA on yeast and mouse networks.

Composition of the Combined Network

Figure 3.9 shows the average weight assigned to each network type for yeast and mouse networks. It is interesting to observe certain consistent trends that are present in both yeast and mouse. For example, in general, BP composite networks have a higher contribution from shared phenotype data (e.g. networks derived from OMIM disease data) and MF composite networks have a higher contribution from shared protein domain data. In addition, as expected, co-expression data are more informative of BP than the MF function. In Figure 3.9, the largest variation in weight between organisms is in the "other" category depicting the data types that were not represented in all three benchmarks. In yeast this category contains co-localization and co-regulation networks, in mouse it contains phylogenic profiling data. For yeast, the majority of the network weight in the "other" category is due to the high, consistent weight placed on protein co-localization data.

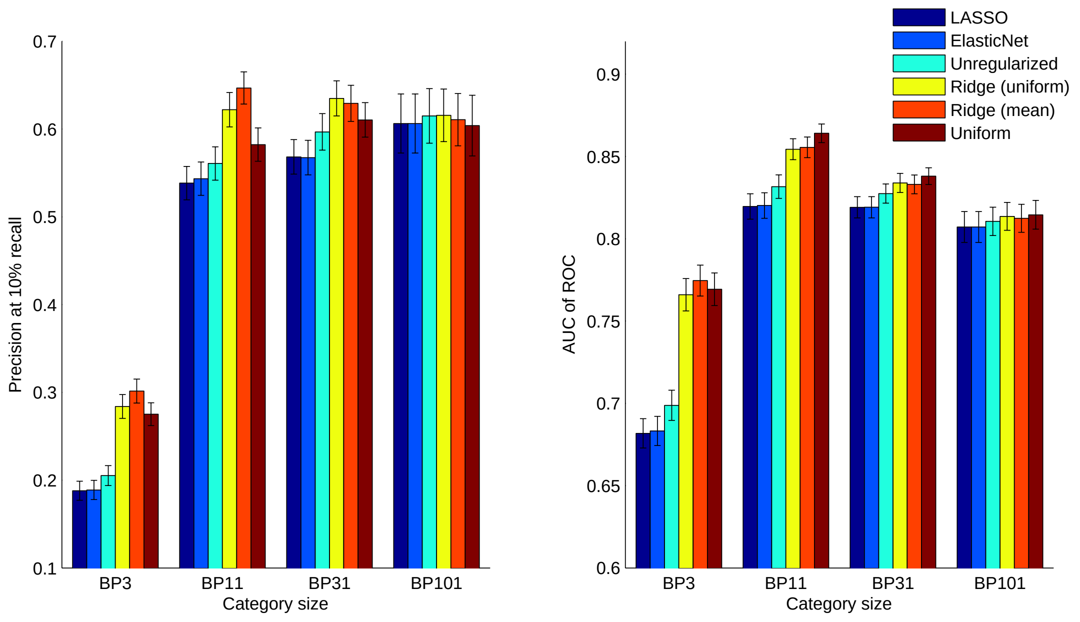


Figure 3.10: Comparison of performance of ridge with uniform- and mean-prior, Lasso, elastic net, and unregularized regression in constructing the combined network. The performance is shown in terms of both precision at 10% recall (left) and the area under the ROC curve (right). Error bars represent the standard error.

3.4.3 Predicting Gene Function with Limited Annotation

In the last section, we showed that when there are only a few positive examples available (e.g. when predicting GO terms which have fewer than 30 annotations) we can greatly improve the performance of GeneMANIA by using ridge regularization with mean-prior. In this section, we extensively evaluate several other regularization procedures, along with Simultaneous Weights (SW), for predicting gene function on several benchmark datasets.

Effect of Regularization

In this section, we compare the performance of unregularized regression with regression that is regularized using Lasso, ridge regression with mean- and uniform-prior, elastic net regularization, and an unweighted network combination. The comparison is conducted on an extended yeast benchmark dataset (Yeast44). We evaluate the performance on all GO terms (downloaded Jan 2007) with 3 to 300 annotations (a total of 1,188 GO categories) and report the performance using 3-fold cross-validation.

Figure 3.10 summarizes the performance of gene function prediction using each method for four evaluation categories (BP GO terms which have [3-10], [11-30], [31-100], and [101-300] positive annotations). Performance is measured in terms of precision at 10% recall and AUROC. In ridge with mean-prior, we set the prior on each network's weight to the average weight that network received in predicting all 1,188 GO biological process categories with 3-300 annotations. We used the LARS [25] algorithm to solve for the Lasso and elastic net solutions; we set the number of positive coefficients using F-statistics [35]. For elastic net, we use cross-validation to select the setting of α_2 from the set [1e-8, 1e-6, 1e-4, 1e-2, 1e-1, 1] that results in the highest AUROC.

Figure 3.10 shows that ridge with mean-prior performs considerably better than the other regularization methods. Consistently with previous studies [54, 97], this figure shows that an unweighted network combination often performs as well as or better than an optimized combination of networks in terms of ROC when there are no irrelevant or redundant networks. This figure also shows that unregularized linear regression performs as well as or better than Lasso, ridge, or elastic net regularization whereas ridge with a prior results in a better performance overall. However, as expected, we see that the performance of unregularized regression improves with increasing number of positives and thus it is more appropriate to use function-specific weighting in such instances.

One explanation for the observed trend in Figure 3.10 is that regularization methods that shrink the network weights toward zero are too selective and often identify only a few relevant networks. For example, on average 45% (20/44), 54% (24/44), and 95% (42/44) of the networks are assigned a non-zero weight using Lasso, unregularized linear regression, and ridge with mean-prior, respectively. Note that the best-performing networks on their own are significantly worse than the combined data (Figure 3.11). As shown in Figure 3.11, the combined network (constructed using ridge with mean-prior),

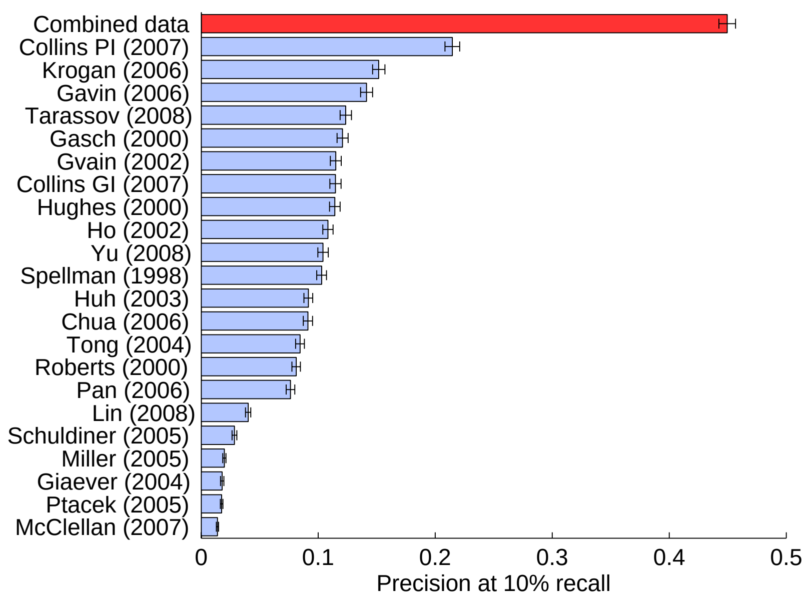


Figure 3.11: The Yeast44 benchmark consist of networks constructed from 22 unique publications (see Table 6.5 in Appendix A). This figure shows the performance of a network constructed from each unique publication, in predicting gene function in terms of precision at 10% recall. The performance of the combined network is shown in red. We construct the combined network using ridge with mean-prior. Error bars represent the standard error.

results in an average precision of 0.48 whereas the best performing network on its own (Collins (PI) 2007 [17]) results in a much lower average precision of 0.22.

Improving the Performance with Simultaneous Weights

As we showed in the previous section, adding ridge with mean-prior regularization to the network integration component of GeneMANIA considerably improves the performance when there is a small number of positive examples. However, to obtain the mean-prior we initially solve a large number of regression problems which increases the computation time of network integration. Instead, SW simultaneously optimizes the network weights to a group of GO categories and doesn't require solving multiple regression problems.

We have investigated four different methods for grouping GO categories for assessing SW: Tree^0, Tree^1, Size, and Clust (see Figure 3.12). In Tree^0 we fit SW to all GO

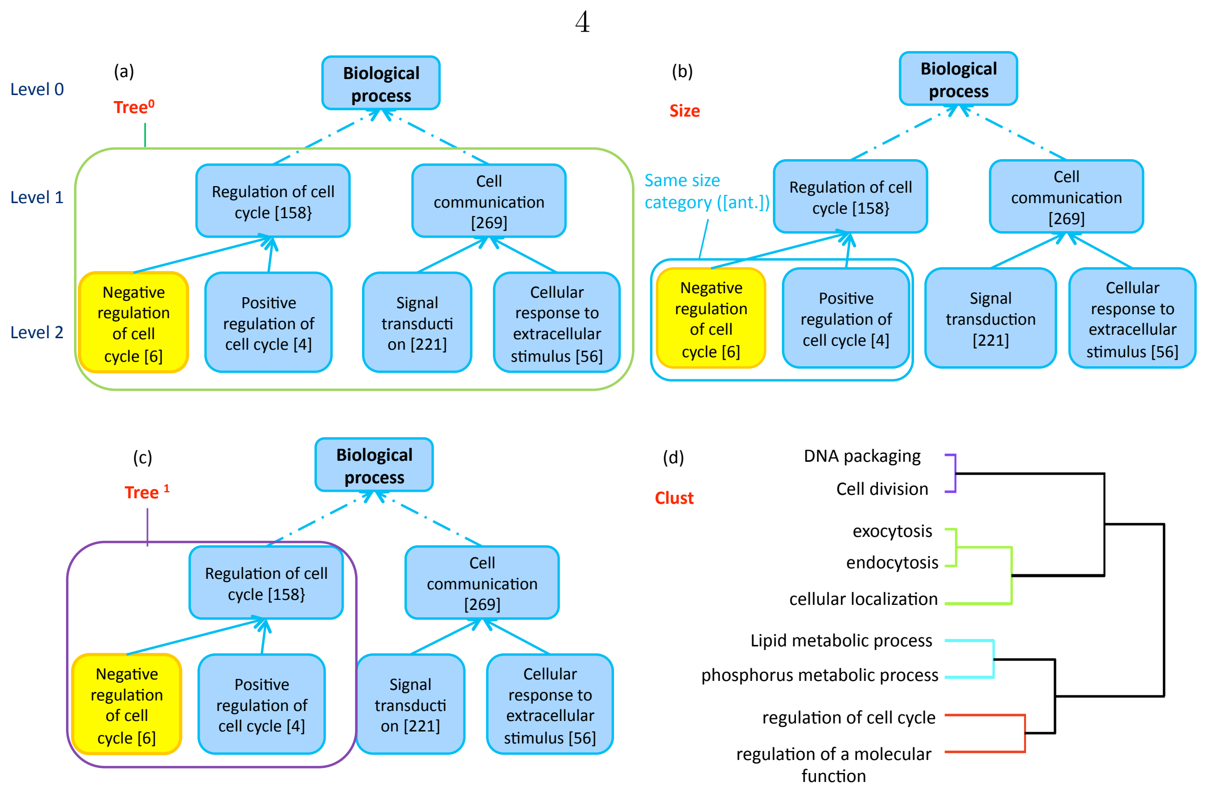

4
(a)
Level 0
Level 1
Level 2
Tree⁰
Biological process
Regulation of cell cycle [158}
Cell communication [269]
Negative regulation of cell cycle [6]
Positive regulation of cell cycle [4]
Signal transducti on [221]
Cellular response to extracellular stimulus [56]
(b)
Size
Same size category ([ant.])
Biological process
Regulation of cell cycle [158}
Cell communication [269]
Negative regulation of cell cycle [6]
Positive regulation of cell cycle [4]
Signal transduction [221]
Cellular response to extracellular stimulus [56]
(c)
Tree ¹
Biological process
Regulation of cell cycle [158}
Cell communication [269]
Negative regulation of cell cycle [6]
Positive regulation of cell cycle [4]
Signal transducti on [221]
Cellular response to extracellular stimulus [56]
(d)
Clust
DNA packaging
Cell division
exocytosis
endocytosis
cellular localization
Lipid metabolic process
phosphorus metabolic process
regulation of cell cycle
regulation of a molecular function

Figure 3.12: We define four different methods for grouping GO categories: (a) Tree^0: all categories in the same hierarchy in GO, (b) Size: all categories in the same GO hierarchy with the similar annotation level where we define 4 annotations levels: [3-10], [11-30], [31-100], and [101-300], (c) Tree^1: all categories in the same hierarchy with the same ancestor at level 1 or lower which has no more than 300 annotations (each term is considered an ancestor of itself), and (d) Clust$^{\#n}$: all categories in the same hierarchy which are clustered together using hierarchical clustering with n clusters (we vary the number of clusters $n = \{3, 10, 20\}$)

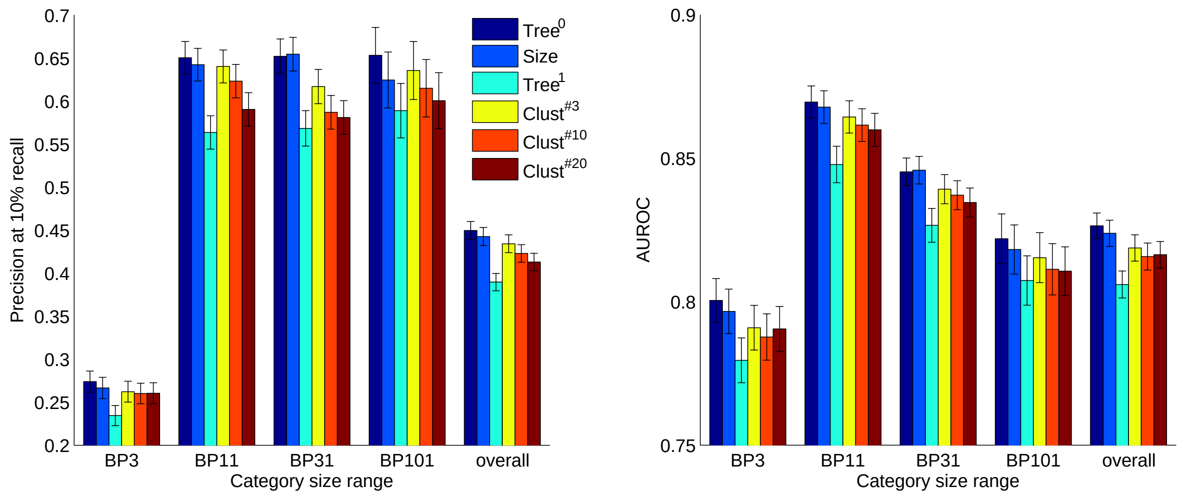


Figure 3.13: Performance of SW with four different groupings of GO categories in predicting BP gene function with 44 yeast networks. The performance is shown in terms of AUC of ROC and precision at 10% recall using 3-fold CV. Error bars represent the standard error.

categories in the same GO hierarchy (e.g. BP) with 3-300 annotations, in contrast, in Tree^1 we fit the weights to all GO categories in the same hierarchy that have the same parent category at level 1 or higher. Note that we only consider GO categories that have less than 300 annotations, therefore, each group in Tree^1 consists of an ancestor with 300 or less annotations and all of its descendants. In Size, we group GO categories based on their number of annotation and hierarchy; for example, we fit one set of weights to all BP categories which have 3-10 annotations. For the Clust method, we use hierarchical agglomerative clustering (single linkage) with Pearson Correlation Coefficient (PCC), of binary vectors which represent the gene annotated to categories, as the similarity metric to cluster GO categories. We investigate three different clusterings with increasing number of clusters $n = \{3, 10, 20\}$. Note that we only consider GO categories with 3-300 annotations; this is because GO categories with fewer annotations have too few examples for training and larger GO categories are too general. Once we compute the network weights based on a group of categories, we construct one composite network and use it to predict all categories in the given group.

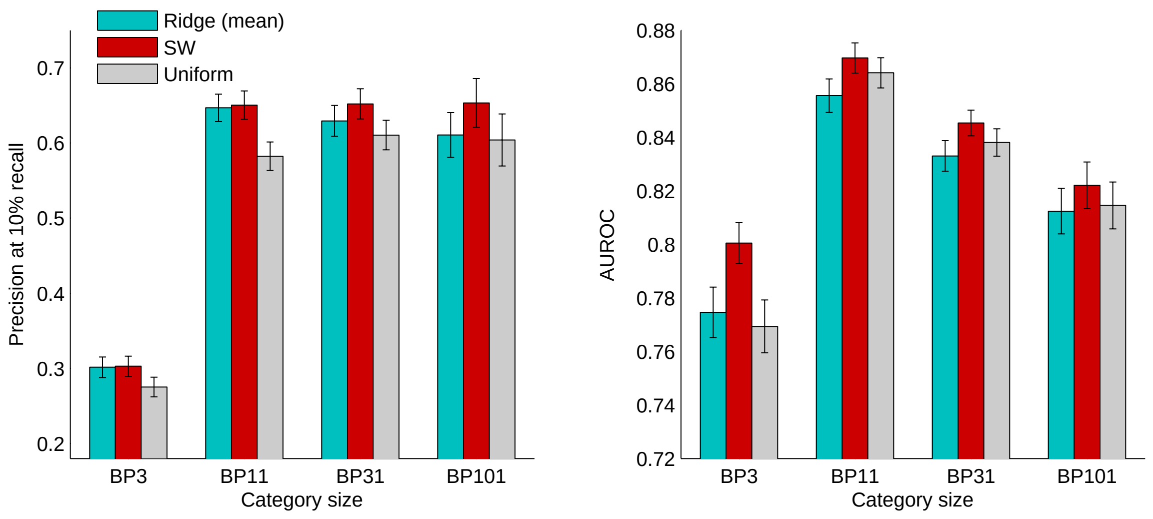


Figure 3.14: Comparison of SW with mean-prior and uniform weighting on large yeast benchmark networks (Yeast44). Error bars represent the standard error.

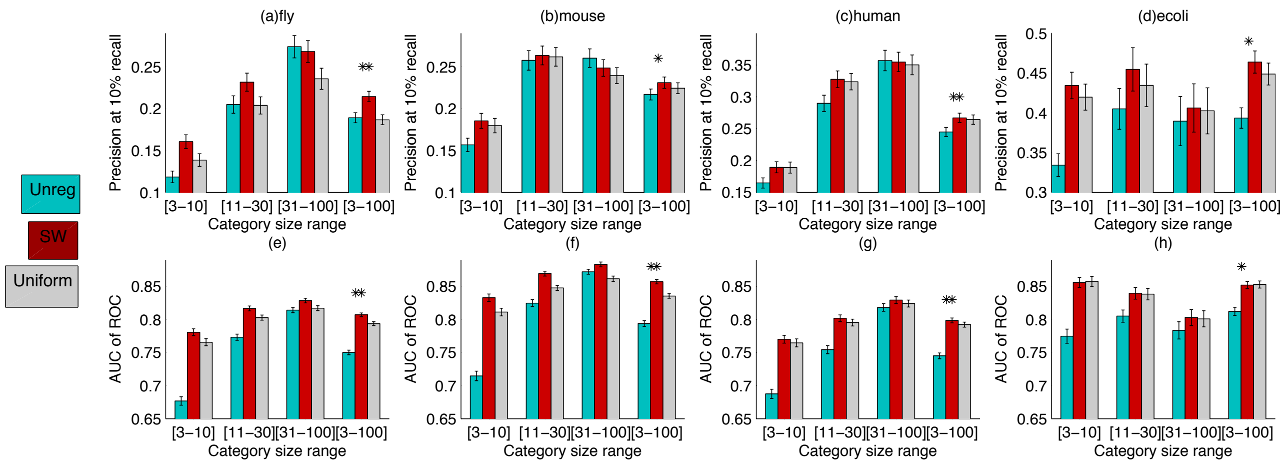


Figure 3.15: Comparison of performance of unregularized linear regression (Unreg), SW, and a fixed uniform combination of networks in predicting gene function in fly ((a) and (e)), mouse ((b) and (f)), human ((c) and (g)), and E. coli((d) and (h)). The bars show average performance in BP categories with [3-10] (n=1,101 for fly, 952 for mouse, for 1,188 for human, 528 for E. coli) [11-30](n=668 for fly, 435 for mouse, 510 for human, 177 for E. coli), [31-100](n=426 for fly, 239 for mouse, 254 for human, and 104 for E. coli) and [3-100] (overall). Error bars show the standard error. Stars indicate significant different in overall performance ([3-100] category size range) using paired Wilcoxon signed rank test with a Bonferroni correction: two stars indicate that SW performs significantly better than both of the other methods and one star indicates that the differences were significant only between SW and unregularized.

We have compared the performance of composite networks constructed by SW when using the above four groupings of GO categories (see Figure 3.13). As shown, the various versions of SW perform similarly, however, SW-Tree^0 slightly out-performs the rest. In addition, Figure 3.13 shows that as the grouping of GO categories becomes more specific (for example with Tree^1 and Clust$^{\#n}$), the predictive performance of SW decreases. In Tree^1, each group consists of the ancestor at 300-annotation level with all of its descendants; within these groups 10 of 1188 categories were singletons (did not have any descendants or ancestors which have 300 or fewer annotations) and 414 of 1188 categories were placed in a group with 10 or more categories. If we remove these singleton categories the performance of Tree^1 is still lower than that of Tree^0 (average area under the ROC curve of 0.8067 for Tree^1 compared to 0.8273 for Tree^0).

Figure 3.14 shows the performance of SW (with Tree^0) compared to ridge with mean prior and a uniform combination. As shown, SW improves the performance of ridge on most evaluation categories. We also investigated the performance of unregularized linear regression, SW, and uniform network weights on fly, mouse, human, and E. coli networks in all GO categories which have between 3-100 annotations (2,195 for fly, 1,626 for mouse, 1,952 for human, and 809 for E. coli). Figure 3.15 summarizes the performance in terms of area under the ROC curve and precision at 10% recall in the four species. As shown, SW is significantly better than unregularized linear regression in the overall category for fly, mouse, human, and E. coli in terms of AUC of ROC. As well, SW is significantly better than uniform and unregularized linear regression in terms of precision in fly and human. In mouse, SW significantly outperforms unregularized linear regression in terms of precision. We note that the human networks are sparser than those of the other organisms, which makes it hard to assign accurate network weights (mean number of interactions is 391,240 in human networks compared to 1,011,400 in mouse) which may explain the smaller (but significant) improvements of SW compared to uniform weights. As well, we note that the performance of uniform weights tends to degrade as the number of networks increases—this is because of the abundance of gene expression datasets and correspondingly large number of co-expression networks. For example, out of the 38 networks for fly, 32 are co-expression networks. By not accounting for redundancy, the performance of uniform weights is significantly worse than that of SW.

3.4.4 Example Biological Application

In the previous sections, we have focused on evaluating GeneMANIA on various benchmarks and settings. As we have shown, GeneMANIA can be used "on-demand" to make accurate, up-to-date predictions about gene function. In particular, in Section 3.4.1, we presented an example application of GeneMANIA to predict genes that are involved in the process of Nucleosome Remodeling. In addition to facilitating fast and accurate prediction of gene function, the GeneMANIA framework allows for three other types of analysis that are useful for biologist: (a) evaluating the "uniqueness" of the functional interactions that are derived from different genomics and proteomics studies, (b) comparing different data types in their predictiveness of gene functions of interest and (c) identifying the types of gene functions that are accurately predicted by a dataset of interest. In this section, we briefly detail the first application.

We can evaluate the uniqueness of a dataset by performing a leave-one-out error analysis, where we perform gene function prediction with all except one dataset. In particular, leaving out a dataset that provides unique information results in more error compared to leaving out datasets that are redundant in the context of the other existing datasets. This analysis allows biologist to evaluate the novelty of a new dataset in the context of all existing datasets.

In collaboration with the Boone lab at University of Toronto (department of Molecular Genetics), we used the GeneMANIA framework to assess the amount of unique information that a new genetic interaction dataset, produced by the Boone Lab, contributed to the existing knowledge about functional relationships between genes. In particular,

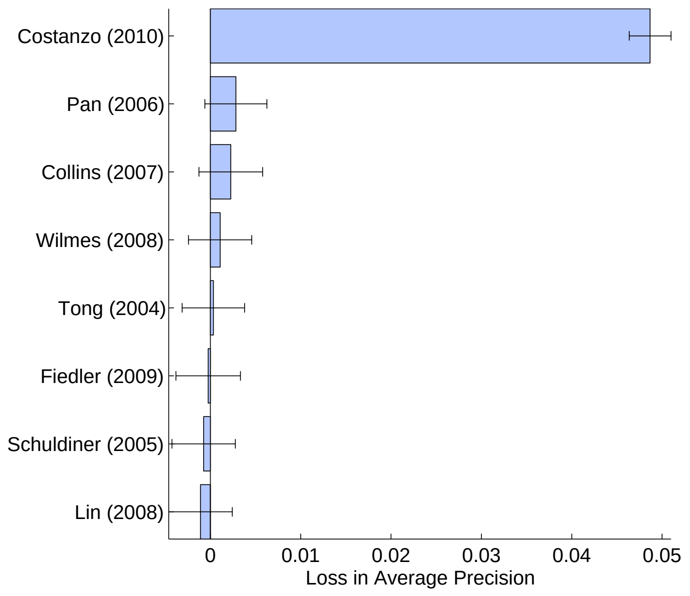


Figure 3.16: Loss in average precision as a result of leave-one-out analysis of various genetic interaction datasets.

we recorded the average precision in predicting all GO biological process categories in yeast with 3-300 annotations in cross-validation using eight available genetic interaction datasets. We then removed each dataset in turn and recorded the resulting difference in precision (Figure 3.16 (results from [19]). In this way, we quantified the average loss in precision in re-producing the known functional relationships in the absence of each genetic interaction dataset. Our results showed that this new dataset (i.e. Costanzo (2010)) provides unique information, resulting in a drastic improvement in predicting gene function.

3.5 Summary

In this chapter, we have presented GeneMANIA, which consists of a constrained, regularized linear regression for combining multiple networks, and a label propagation algorithm for predicting gene function from the combined network. We have shown that GeneMANIA is as accurate as, or more accurate than, leading gene function prediction algorithms on yeast and mouse despite requiring orders of magnitude less computation time than many of the alternatives. We achieve the highest accuracy using a version of our algorithm that requires between 10 and 15 seconds computation time on a desktop computer. Consequently, we have demonstrated that it is possible to design a gene function prediction algorithm that performs on-demand function prediction with the most up-to-date annotation list and data sources available, while achieving the same or better accuracy as leading algorithms.

For a given gene function, the output of GeneMANIA is a vector of discriminant scores $\mathbf{f} \in [-1, 1]^n$, where f_i represents the likelihood that gene i is involved in the function of interest. Below, we will describe few approaches for computing confidence scores or uncertainty for these discriminant scores.

In Section 2.6.2, we described the probabilistic formulation of LPA and showed that the discriminant scores can be obtained by performing MAP estimation in Gaussian Markov Random Fields (GMRF), where the discriminant scores are the posterior mean of a Gaussian distribution. Using the GMRF formulation, we can further obtain the posterior variance for each discriminant score. In particular, we showed that $\mathbf{f} = (I + \lambda L)^{-1}\mathbf{y}$ where $\mathbf{f}$ is the mean of the following Gaussian distribution: $N((I + \lambda L)^{-1}\mathbf{y}, (I + \lambda L)^{-1})$. Therefore, $M = (I + \lambda L)^{-1}$ is the posterior covariance matrix and M_{ii} represents the posterior variance of f_i. One use of the posterior variance could be as a confidence interval that indicated our uncertainty in f_i. We could, for example, use these confidence intervals to convert the ranking of the f_i value's into a partial order of f_i's that we are certain are larger than others.

One simple method to convert the discriminant scores to probabilities is to use Platt's method ([78]). In particular, Platt's method converts discriminant scores $\mathbf{f}$ to probabilities using logistic regression: $p(y_i = 1|f_i) = 1/(1 + e^{-(b_0+b_1 f_i)})$ where the parameters b_0 and b_1 are fitted using cross-validation. To do so, we leave out a portion of positive and negative examples and compute $\mathbf{f}$. We then fit b_0 and b_1 using the left out examples with the maximum likelihood method.

We can also obtain empirical p-values through permutation tests. To do so, we permute the initial label vector $\mathbf{y}$ multiple times and compute the discriminant scores for the test examples. Using the obtained discriminant scores on permuted data, we then construct a null distribution for the discriminant scores. We can then obtain a p-value for each f_i as the fraction of the discriminant scores in the null distribution that are equal or larger than f_i.

Chapter 4

Incorporating Ontology Structure into Predictions

4.1 Introduction

As we discussed in Chapter 2, hierarchical gene classification schemes, such as Gene Ontology (GO), organize gene function categories as a directed acyclic graph (DAG) in which categories describing broader functions (e.g. eye development) are ancestors of those describing more specific functions (e.g. eye photoreceptor cell differentiation). In this chapter, we consider ways to incorporate the hierarchical organization of function categorization schemes such as GO when making predictions about gene function.

Curators annotate genes by associating them with the most specific category (GO term) supported by the available data. Often genes are annotated using internal nodes of the DAG because there is insufficient evidence to annotate genes in the most specific, i.e., leaf, categories. For example, a mouse gene can be annotated as being involved in development if mice with defective copies of that gene die as embryos, before further investigations are done to determine whether the gene functions in, e.g. eye, heart, or brain development. These internal node annotations can provide helpful hints when classifying genes in descendent categories, so long as the classification algorithm is able to incorporate prior knowledge about the hierarchy.

Here, we introduce two new classification methods that leverage DAG-based categorization hierarchies. Both of our algorithms extend the label propagation algorithm [107, 109]. Our interest in this algorithm stems from its success at predicting gene function compared with other binary and hierarchy-based classification schemes [76, 60, 97]. Our first method, which we call *Hierarchical label propagation (HLProp)*, replicates the affinity (similarity) network for each category and then links the nodes representing the same gene in parent and child categories, thus ensuring that the discriminant scores of a gene in related function categories also remain close. By applying the label propagation algorithm to this new, much larger (though sparsely-connected) network, we can perform multi-label classification efficiently by solving a linear system of equations. We also describe a second method, *Hierarchical label bias (HLBias)*, that uses the GO hierarchy to set label biases of genes with annotations in internal category nodes. This second approach builds on the previous work of [26] which used the structure of the GO hierarchy to define positive and negative examples for a given category of interest.

This chapter is organized as follows. First we introduce our two proposed methods, HLBias and HLProp, and describe two variants of HLProp. Next, we evaluate these methods on the MouseFunc benchmark networks (see Section 3.2) using both a "test set" and a "novel set". In particular, we compare the performance of HLBias and HLProp with the regular version of label propagation, which doesn't take advantage of the hierarchical structure of GO, and Isotonic Regression (IR) (see Section 2.8.2), one of the state-of-the-art *reconciliation* methods [69] that considers the DAG hierarchy. Finally, we conclude this chapter with a discussion and summary of our results. The results presented here appear in [58].

4.2 Methods

We assume that we are given a network represented by a symmetric affinity matrix, $W = W^{\mathsf{T}}$, over n genes with $w_{ij} \geq 0$; W can be a single network or constructed from multiple networks (as described in Chapter 3). We represent the labels for multiple GO terms (categories) with the matrix $Y_{n \times d} = \{0, 1\}^{n \times d}$, where d is the number of categories; $y_{ic} = 1$ if gene i is annotated to category c. We represent the hierarchy structure with a matrix $H_{d \times d}$ where $h_{ij} = 1$ if category i is a parent of category j in the GO hierarchy. The column c of Y, denoted by $\mathbf{y}^{(c)}$, represents the labels for category c. Below, we first describe HLBias and then we describe HLProp and two of its variants.

4.2.1 Hierarchical Label Bias

HLBias builds on the previous work of [45] and [26]. In particular, King and colleagues [45] used a gene's annotations as its feature vector for predicting additional annotations for the given gene. Eisner and colleagues [26], used the structure of the GO hierarchy to define appropriate negative examples for predicting a given category: the appropriate negative examples for a given category c were deemed to be genes which have no annotations in descendants or ancestral categories of c.

Recall that the solution to the label propagation algorithm (LPA) can be interpreted as the MAP estimate in an appropriate Gaussian Markov Random Field (see Section 2.6.2). This interpretation suggests that the initial label y_{ic} reflects our prior bias that gene i is annotated to category c (see Section 2.6.2). Accordingly, in HLBias, we use a gene's previous annotations to estimate our prior bias that it will be annotated to a given category of interest.

In our setting, when predicting category c, we first use as negatives all genes that are annotated (positive examples) to any sibling category of c. We assign this negative label because genes are rarely annotated in more than one child of the same parent category. For other genes i with an annotation in an ancestral category a of c, we set $y_{ic} = 2 \times \frac{n^+_{ac}}{n^+_a} - 1$ where n^+_a is the number of positive examples in category a and n^+_{ac} is the number of positive examples in category a that were also annotated in category c; this initial label bias is proportional to the probability of a gene being annotated to category c given its annotation in category a. For a gene i with multiple annotations, we set y_{ic} to its mean value; for example, if i is annotated to categories a and b, we set its initial label bias for category c to the mean of $2 \times \frac{n^+_{ac}}{n^+_a} - 1$ and $2 \times \frac{n^+_{bc}}{n^+_b} - 1$. Having set these label biases, we then solve for discriminant scores by using label propagation to predict each category independently.

4.2.2 Hierarchical Label Propagation

Hierarchical Label Propagation (HLProp) extends the label propagation algorithm to include the hierarchical organization scheme. As we described in Section 2.6.2, we can determine the solution to the label propagation problem by solving a convex optimization problem. In particular, we can write the corresponding convex optimization problem (given by Equation (2.6)) in scalar form as follows:

$$\mathbf{f}^* = \operatorname*{argmin}_{\mathbf{f}} \sum_{i}^{n} (f_i - y_i)^2 + \lambda \sum_{i,j}^{n} w_{ij} (f_i - f_j)^2 \tag{4.1}$$

$y_i = \{-1, k, +1\}$ is the label of the node (gene) i, where positive genes are labeled as 1, negative genes are labeled as -1, and unlabaled genes are labeled as $k = \frac{n^+ - n^-}{n}$ (see Section 3.3.2). In HLProp we solve for the discriminant scores for all d categories (GO terms) simultaneously while ensuring that nearby categories have similar discriminant scores. To do so, we solve the following problem:

$$F^* = \underset{F}{\text{argmin}} \sum_i^n \sum_c^d (y_{ic} - f_{ic})^2 + \tag{4.2}$$
$$\lambda \sum_c^d \sum_{i,j}^n w_{ij}(f_{ic} - f_{jc})^2 + \gamma \sum_i^n \sum_{c,m}^d h_{mc}(f_{im} - f_{ic})^2$$

where $F = [\mathbf{f}^{(1)}, ..., \mathbf{f}^{(d)}]$ and h_{mc} indicates whether m and c are directly connected in GO, and λ, γ are the regularization constants. Without the third term (by setting $\gamma = 0$), equation (4.2) corresponds to solving d independent binary classification problems. The third term encourages the discriminant values of a gene in two related function categories to be similar to each other (see Figure 4.1 for an example).

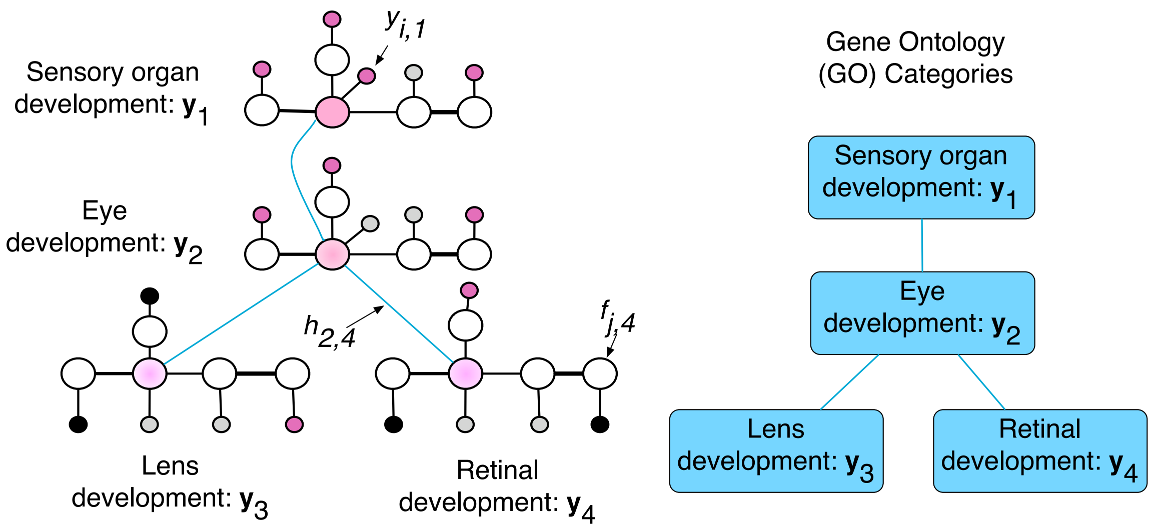
As with regular LPA, we set $y_{ic} \in \{-1, k, +1\}$, where negatives for category c are represented by -1, positives for category c are represented as +1, and unlabeled nodes are represented as $k = \frac{n_c^+ - n_c^-}{n}$. When making prediction for the "test set", we assume non-positives, non-test genes are negatives. For the "novel set", for each category, the initial labels are $\{-1, 1\}$ for the positive and non-positive genes.

In this work, we use the GO hierarchy to define H. GO is a DAG; however, we treat GO as an undirected graph. In particular, $h_{cm} \in \{0, +1\}$ represents the parent-child relationships in GO: we set $h_{cm} = h_{mc} = 1$ if m is a parent of c or vice versa. In addition, in our experiments we set λ and each γ to a fixed value of 1 and so we drop these constants from our subsequent equations.

Optimization

We can solve for F^* by solving the following problem:

$$\begin{aligned} F^* &= \underset{F}{\text{argmin}} \ \text{trace}(F^\mathsf{T} F - 2F^\mathsf{T} Y) \\ &+ \ \text{trace}(F^\mathsf{T} L F) \ + \ \text{trace}(F G F^\mathsf{T}) \end{aligned} \tag{4.3}$$



$y_{i,1}$
Sensory organ development: $\mathbf{y}_1$
Eye development: $\mathbf{y}_2$
$h_{2,4}$
$f_{j,4}$
Lens development: $\mathbf{y}_3$
Retinal development: $\mathbf{y}_4$
Gene Ontology (GO) Categories
Sensory organ development: $\mathbf{y}_1$
Eye development: $\mathbf{y}_2$
Lens development: $\mathbf{y}_3$
Retinal development: $\mathbf{y}_4$

Figure 4.1: A graphical example of our model. In the left figure, there are four identical networks over four genes (nodes); the associations between different genes is depicted by black edges. The colors of the smaller nodes attached to each gene represent the initial label of the gene. Pink indicates +1 and black -1; grey indicates the initial label bias of the unlabeled genes. If we wish to predict which genes are involved in *eye development*, we need to consider other related categories (as shown on the right). In our modification, we introduce an edge between the same gene (blue edges) in the different networks (in this figure, we have only shown blue edges for one gene); these edges will encourage the discriminant value of the same gene (depicted as the color of the bigger nodes) in related categories to be similar.

where G is the Laplacian of H, i.e. $G = V - H$ where V is a diagonal matrix with $v_{cc} = \sum_m^d h_{mc}$.

Taking the derivative of (4.3) with respect to the matrix F, we get the matrix equation $(I + L)F + FG = Y$. Equivalently, we can find F by solving a large sparse linear system: $A \times (\text{vec}(F)) = \text{vec}(Y)$, where $\text{vec}(Y)$ is an operator that stacks the columns of Y atop of each other, $A_{(n\times d)\times(n\times d)} = (I_{d\times d} \otimes (I + L) + G \otimes I_{n\times n})$, and $\otimes$ denotes the Kronecker matrix product. As an example, the matrix A that corresponds to the example in Figure 4.1 can be represented as:

$$A = \begin{pmatrix} (2I + L) & -I & 0 & 0 \\ -I & (3I + L) & -I & -I \\ 0 & -I & (2I + L) & 0 \\ 0 & -I & 0 & (2I + L) \end{pmatrix}$$

In general, A can be represented as a block matrix with diagonal blocks $A_{ii} = (I + L + v_{ii}I)$ and non-diagonal blocks $-h_{ij}I$.

When H is symmetric, then A is also symmetric. Furthermore, since A is diagonally dominant with positive diagonals, A is symmetric positive definite (SPD) and thus invertible. However, for large d (number of gene function categories) and n (number of genes), constructing A may be infeasible. Instead, we can solve for the $\mathbf{f}^{(c)}$'s iteratively: given all $\mathbf{f}^{(c)}$'s for $c \neq m$, we can solve for $\mathbf{f}^{(m)}$ by solving the system of linear equations: $(I + L + v_{mm}I)\mathbf{f}^{(m)} = \sum_c^d h_{cm}\mathbf{f}^{(m)} + \mathbf{y}^{(m)}$. In our setting, problem (4.3) is convex and we can calculate F^* by iteratively updating $\mathbf{f}^{(m)}$'s; we have empirically observed that we need 10 or fewer iterations to solve each $\mathbf{f}^{(m)*}$ when there are approximately 50 GO categories that are related to each other.

4.2.3 Down- or Up-Propagation

In HLProp, we assume that the discriminant scores of a parent-child category pair $\mathbf{f}^{(c)}, \mathbf{f}^{(\pi_c)}$ should be similar to each other. In a simpler model, we can only restrict the discriminant scores of the child to be similar to the parents and not *vice versa*. In this setting, we can first solve for the $\mathbf{f}^{(r)}$, where r corresponds to the category that is at the root of the hierarchy, and then solve for each $\mathbf{f}^{(c)}$, where c is a child of r; continuing this process, we can iteratively solve for all children of c and so on. Conversely, we can flip the edges so that the leaves of the hierarchy become the roots, we can then solve $\mathbf{f}^{(l)}$'s for the leaf nodes l and propagate these up to solve for $\mathbf{f}^{(r)}$. As a heuristic approximation to HLProp, we solve for $\mathbf{f}^{(c)}$'s in two ways: (a) first solving for $\mathbf{f}^{(r)}$ and then propagating down to solve for all the descendant nodes (Down-Propagation) and (b) first solving for all the leaf nodes $\mathbf{f}^{(l)}$ and then propagating these up to solve for all ancestor nodes $\mathbf{f}^{(c)}$'s (Up-Propagation).

4.3 Results

We perform our experiments using the MouseFunc benchmark networks [76] (see Section 3.2). We evaluate our methods, and compare their performance to regular LPA (as a baseline) and IR (see Section 2.8.2), by using a "test set" (3-fold cross-validation) and a "novel set" derived from GO BP annotations. For cross-validation, we perform our analysis on all 2,634 GO BP categories (downloaded Sept. 2007) with three to three hundred annotations. The novel set consists of a set of genes that acquired new annotations in GO from September 2007 to September 2008. Below, we first show the performance on the "test set" and then focus our analysis in predicting the "novel set".

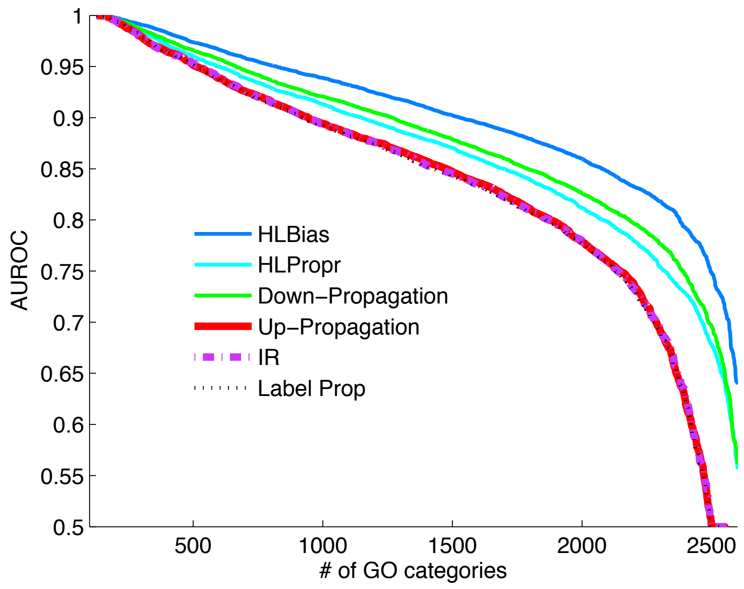


Figure 4.2: Cumulative performance of various methods in predicting the function of test genes (cross-validation) in 2,634 GO categories in terms of AUROC.

4.3.1 Predicting Test Genes

Figure 4.2 shows the cumulative distribution of the AUROC of the Hierarchical label bias, HLProp, Down- and Up-Propagation, IR (as used in [69]) and regular label propagation. See Section 2.8.2 for a description of IR. Figure 4.3 summarizes the average performance in 4 evaluation categories: BP categories with [3-10] annotations (BP3), [11-30] annotations (BP11), [31-100] annotations (BP31), and [101-300] annotations (BP101).

HLProp, HLBias, and Down-Propagation considerably improve the performance of gene function prediction. Specifically, despite being the simplest method, HLBias achieves the best overall performance in terms of AUROC. Note that in the cross-validation setting, for HLBias, the test genes are labeled as unknowns and the initial label of other non-positive genes is set according to their previous annotations in the GO hierarchy. One explanation for the better performance of HLBias compared to HLProp is that, in using HLBias, genes that have an incomplete annotation in an ancestral category more directly influence the discriminant scores of their nearby genes. This is because the initial

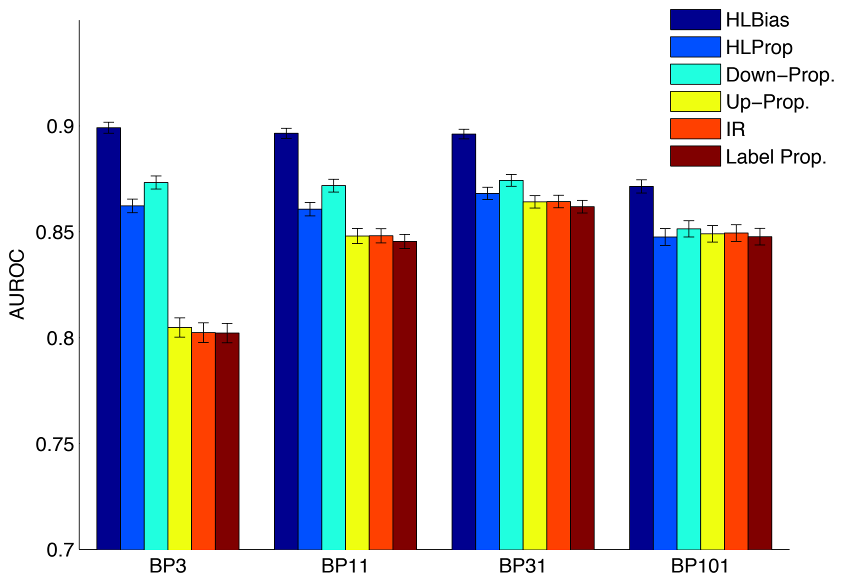


Figure 4.3: Performance of various methods in predicting the function of test genes (cross-validation) in 2,634 GO categories in terms of AUROC. The performance is shown in four evaluation categories: average performance on categories with [3-10] annotations (BP3), [11-30] annotations (BP11), [31-100] annotations (BP31), and [101-300] annotations. The error bars represent the standard error.

label bias of a gene essentially needs to be propagated through a minimum of two edges to affect the label bias of a test gene; in HLProp the incomplete annotation information needs to be propagated through a minimum of three edges to affect the discriminant score of a test gene (see Figure 4.4 for a pictoral description). Down-Propagation performs very similar to HLProp, suggesting that propagating information down the hierarchy is the most informative component in HLProp. In addition, we observed that IR and Up-Propagation do not significantly improve performance. This result is consistent with the observations in [69] that most reconciliation methods often perform similarly to the baseline of independent predictions.

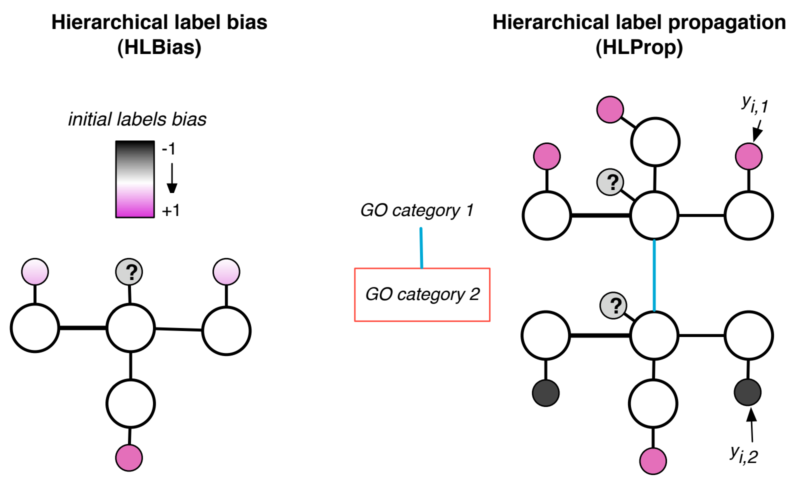

Hierarchical label bias (HLBias)
initial labels bias
-1
+1
?
Hierarchical label propagation (HLProp)
GO category 1
GO category 2
$y_{i,1}$
$y_{i,2}$

Figure 4.4: An example illustrating the difference between HLBias and HLProp in assigning discriminant scores to a test gene. The test gene is depicted by the node whose initial label bias is a question mark. In the right figure, the nodes in the same column depict the same gene in different categories; only one of the five blue edges representing each h_{ac} is shown. In HLBias, when predicting GO category 2, the two neighboring nodes of the test gene have a more positive label bias. In HLProp, the previous annotations need to be propagated through more edges to affect the discriminant score of the test gene.

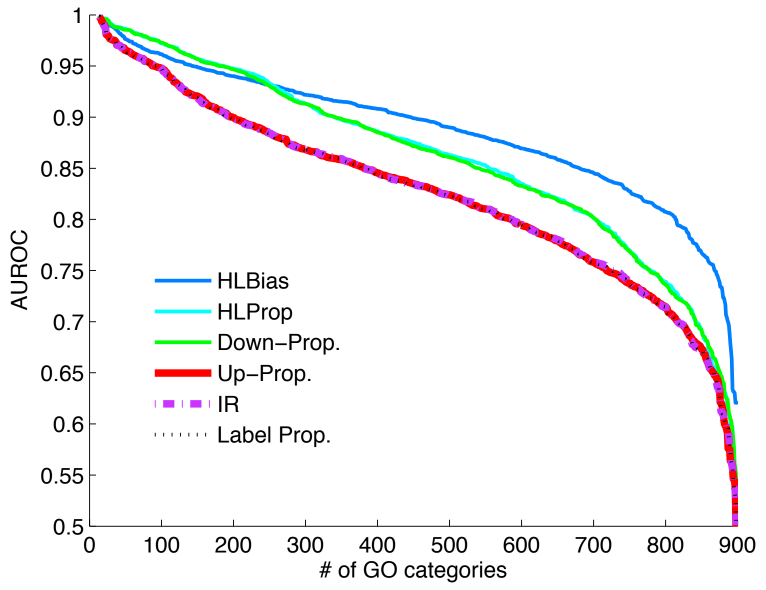


Figure 4.5: Cumulative performance of HLBias and HLProp on the novel set of 903 categories that acquired 3 or more annotations in terms of AUROC.

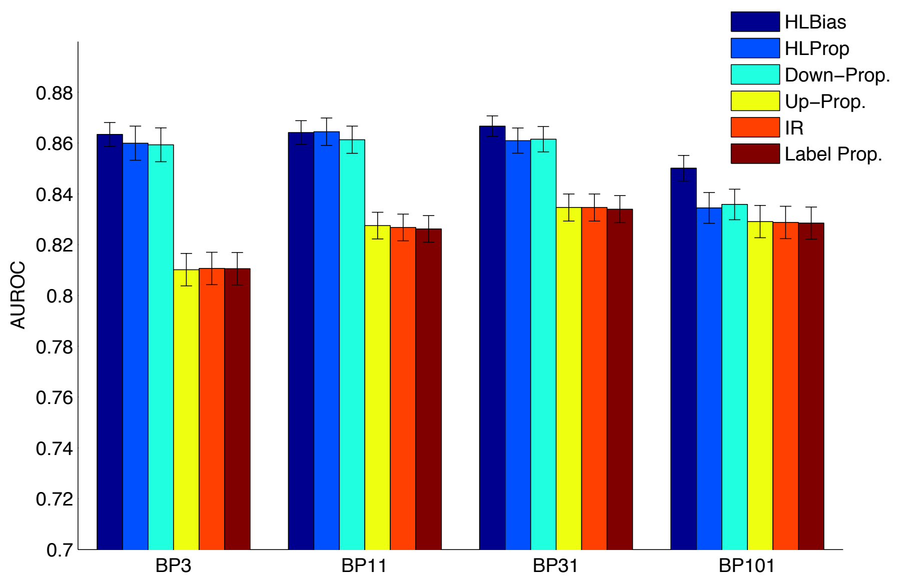


Figure 4.6: Average performance of HLBias and HLProp on the novel set in terms of AUROC. The performance is shown for four evaluation categories: average performance on categories with [3-10] annotations (BP3), [11-30] annotations (BP11), [31-100] annotations (BP31), and [101-300] annotations. The error bars represent the standard error.

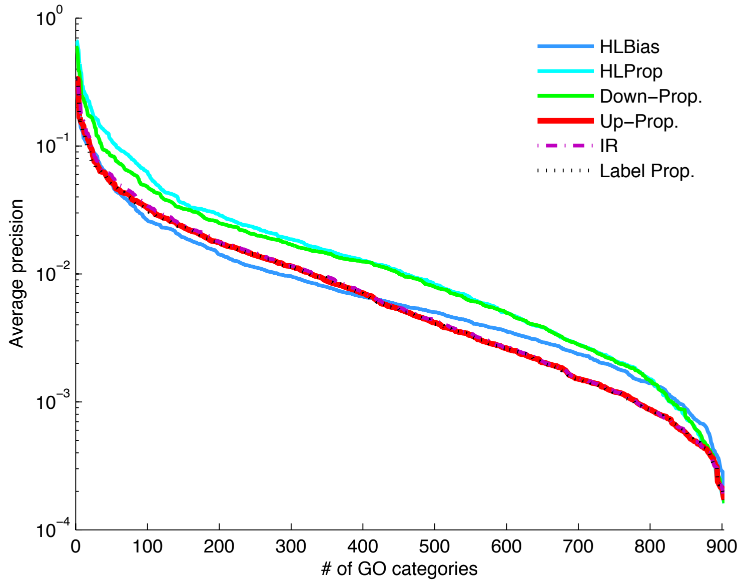


Figure 4.7: Comparison of performance in terms of average precision when predicting novel genes in 903 GO categories that acquired three or more annotations in a span of one year.

4.3.2 Predicting Novel Genes

Here we report performance on the "novel set". In this setting, to evaluate the performance on a given category, we use newly annotated genes as positives and all other genes (excluding previously annotated genes, that is, those annotated in the 2007 GO file) as negatives. When predicting a given GO category with HLBias, we adjust the initial label of all except the positive genes (which have an initial label bias of +1 according to the 2007 GO file) by using the incomplete annotation information in GO.

Figure 4.5 shows the cumulative performance of each method in predicting novel genes in 903 categories that acquired new annotations in the span of one year. Similarly to the cross-validation results, HLBias, Down-Propagation and HLProp drastically improve on the baseline performance of LPA. To better understand the difference between the various methods, we measured the mean performance in predicting GO categories at four different specificity levels: those with [3-10], [11-30], [31-100], and [101-300] annotations in the 2008 GO file. As shown in Figure 4.6, HLBias performs better than HLProp and Down-Propagation when predicting GO categories with [101-300] annotations. The performance of HLProp, Down-Propagation, and HLBias is similar when predicting GO categories with [11-100] annotations.

In addition to measuring performance in terms of AUROC, we also investigated performance in predicting novel gene functions in terms of average precision. As shown in Figure 4.7, HLProp outperforms all other methods in terms of average precision. Interestingly, in contrast to its performance in terms of AUROC, we observed that on average, the performance of HLBias is not significantly different than the baseline approach (t-test with α=0.05). However, the cumulative performance of HLBias follows the same trend as measured in AUROC or average precision (compare Figure 4.5 and 4.7); HLBias has a lower precision at higher percentiles but higher precision at lower percentiles.

4.4 Summary

Here we have shown that by using the GO hierarchy information directly, either by setting initial label biases using GO or using our formulation of hierarchical label propagation (HLProp), we can significantly improve gene function prediction. On the other hand, our results are consistent with the previous report that reconciliation methods may rarely improve the performance of independent classifiers [69]; in our setting, the reconciliation of independent GO categories preform very similarly to the baseline of *unreconciled* classifications obtained by regular LPA.

In order to be able to solve HLProp efficiently, we ignored the directionality of the GO hierarchy. To do so, we set $h_{mc} = h_{cm}$ if category c is a child of category m. In contrast, the two heuristic variants (Up- and Down-Propagation) only propagate information about discriminant scores in one direction. Our results indicate that propagating information down the hierarchy results in the most gain whereas Up-Propagation does not significantly affect the performance. This result is consistent with that of [69] which found that reconciliation with a Bayesian network model of the GO hierarchy, where the arrows are directed from parents to the child classes performs better than the opposite model where the arrows are directed from children to parents.

Our results on the "novel set" revealed the contrasting conclusions that may result from using AUROC or average precision to evaluate performance in gene function prediction. HLBias results in the overall highest AUROC whereas its performance in terms of average precision is slightly better than the baseline of LPA. This suggest that HLBias has better performance at high recall rather than low recall and that HLBias gains in performance by correctly assigning low discriminant scores to negative examples. Since HLProp (or Down-Propagation) and HLBias tend to improve the performance of regular LPA in different ways, combining the two methods may result in even a better performance in terms of both AUROC and average precision.

Chapter 5

Predicting Binary Node Labels for Very Large Networks

5.1 Introduction

In this chapter, we focus on the general task of predicting binary node labels from an arbitrary network. In particular, given a network over n genes, represented by the matrix $W_{n \times n}$, and a set of positive nodes, our goal is to predict which other nodes are likely to be positives. We denote the initial labels as $\mathbf{y} = \{0, 1\}^n$—here we assume we only have positive and unlabeled nodes. Binary prediction of gene function is one instantiation of this problem. Other instantiations include predicting genes that are involved in a given disease [71], or the subcellular localization of proteins [70], based on their physical interaction networks. Some example problems in other domains include predicting product preferences [73] from social networks where people are connected based on friendship ties, and predicting the *political view* (e.g. liberal or conservative) of blogs based on web links.

In Chapter 3, we have shown that the conjugate gradient (CG) algorithm allows us to obtain the solution to the label propagation algorithm (LPA) in seconds for fairly large networks (e.g. those with 20,000 nodes and 8M edges). In this chapter, we investigate the process of solving the iterative formulation of LPA (described in Section 2.6.2). As we will show, in many real-world networks we can even further improve on the running time required by CG. In fact, we can obtain an accurate approximation to the solution to LPA with fewer than five iterations, each requiring one matrix-vector product.

In addition, our investigation shows that the geometrically decreasing contribution of walks of increasing length, as assumed in LPA (see Section 2.6.2), is not always accurate. As a simple example, in a genetic interaction network, nodes that are connected by a walk of length two are disproportionately more likely to be functionally related than those connected by a walk of length one [19]. Based on these observations, we propose an alternative label propagation algorithm, which we refer to as the *Weighted Proximal propagation Algorithm (WPA)*, for predicting binary node labels in very large networks. As we will show, WPA is faster and results in more accurate predictions in a variety of networks.

In this chapter, we first review three versions of LPA and compare their performance in predicting gene function from protein and genetic interaction networks. We then investigate the performance of iterative LPA with increasing numbers of iterations. Finally, we describe WPA and compare its performance to LPA on a protein interaction network, a genetic interaction network, five social networks, a blogs web-link network, and a patent-citation network.

5.2 Benchmark Networks

We evaluate the performance of LPA and WPA on a protein interaction network (PI), genetic interaction network (GI), five social networks (Facebook), a web-link network (Blogs), and a patent-citation network (Patent), as described below:

- **Yeast Protein interaction network (PI)**. We construct a protein-interaction network from all high-throughput interactions in BioGRID [91] (downloaded April 2010). This network consists of 5,306 nodes and 71,262 edges.

- **Yeast Genetic interaction network (GI)**. We construct a genetic-interaction network from all high-throughput *negative* genetic interactions in BioGRID [91] (downloaded April 2010). This network consists of 4,563 nodes and 152,188 edges.

- **Facebook Networks**. We obtain five Facebook networks from [94]. The nodes in these network represents students and the edges represents friendship ties. The five different networks were obtained from friendships between students at five universities: Caltech (769 nodes and 33,312 edges), Princeton (6,596 nodes and 586,640 edges), Oklahoma (17,425 nodes and 1,785,056 edges), University of North Carolina (UNC) (18,163 nodes and 1,533,600 edges), and Georgetown (9,414 nodes and 851,276 edges).

- **Blogs Network**. We obtain a network of links between political blogs from [1]. This network consists of 1,224 nodes and 33,433 edges.

- **Patent Network**. We obtain patent-citation data from [32]. In this network, nodes represent patents and edges represent a patent citation: two patents are connected if either one cites the other. This network consists of 3M nodes and 18M edges.

For the PI and GI networks, the labels are annotated gene functions. We use 47 non-redundant GO functions (downloaded April 2010) that have more than 30 annotations. These functions were deemed to be a minimal set of non-redundant and informative functions by a panel of biologists [62]. For the Facebook networks, we use genders as labels. For the Blog network, we use political views (liberal or conservative) as labels. For the patent network, we use 381 patent categories as labels.

5.3 Methods

5.3.1 Iterative LPA and Random Walks

In Chapter 2, we described an iterative version of LPA where the discriminant scores $\mathbf{f}$ are obtained using an infinite sum:

$$\mathbf{f} = (1 - \lambda) \sum_{r=0}^{\infty} (\lambda W)^r \mathbf{y}. \tag{5.1}$$

Under the condition that the eigenvalues of W are in $[-1, +1]$ (denoted by $\rho(W) \leq 1$) and $0 < \lambda < 1$, the infinite sum converges and we obtain: $\mathbf{f} = (1 - \lambda)(I - \lambda W)^{-1}\mathbf{y}$.

We consider two different normalizations of W for guaranteeing that $\rho(W) \leq 1$ [16]: asymmetric normalization $P = D^{-1}W$ and symmetric normalization $S = D^{-1/2}WD^{-1/2}$ where D is the diagonal row sum matrix $d_{ii} = \sum_j^n w_{ij}$. Yet, we can obtain a third convergent version of LPA by using the convex optimization formulation without normalizing the Laplacian: $\mathbf{f} = (1 - \lambda)(I + \lambda L)^{-1}\mathbf{y}$ (see Section 2.6.2).

Figure 5.1 shows the performance of the three versions of LPA for protein-interaction (PI) and genetic-interaction (GI) networks. These variants of LPA are obtained by using the symmetrically normalized S (we will refer to this as SymLPA) or the Markov Transition Matrix P (we will refer to this as AsymLPA) or using the (un-normalized) Laplacian L in the convex optimization formulation. As shown, all three methods result in similar performance in terms of AUROC, however, using SymLPA results in better performance in terms of average precision. In the rest of this Chapter, we will use SymLPA $\mathbf{f} = (1 - \lambda)(I - \lambda S)^{-1}\mathbf{y}$ when performing label propagation.

Truncated LPA

To determine the contribution of walks of increasing length to the performance of LPA, we compare the result of truncated LPA: $\mathbf{f} = \sum_{r=0}^{m} (\lambda S)^r \mathbf{y}$ for $m = \{1, 2, ..., 10\}$, with

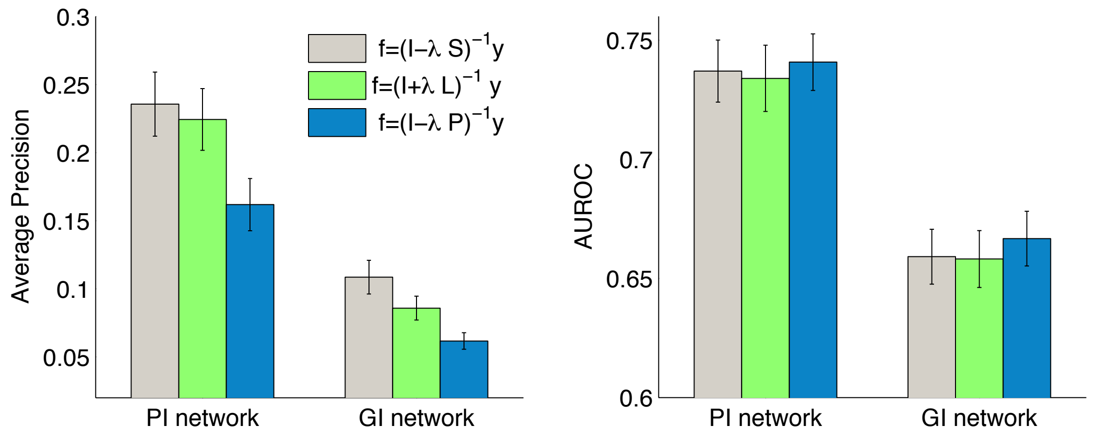


Figure 5.1: Comparison of label propagation with three different normalization methods on the protein-interaction (PI) network and genetic-interaction (GI) networks. The performance is shown in terms of AUROC (right) and average precision (left) on 47 informative and non-redundant GO categories [62]. We determine the optimal setting of parameter λ for each method using cross-validation. Error bars represent the standard error.

the exact solution. When $m = 1$, the truncated LPA corresponds to the *weighted-voting* classifier which only considers direct network neighbors in making predictions. Similarly, when $m = 2$, truncated LPA only considers first- and second-degree neighbors (those that are connected to the positives with a walk of length two or less). In general, when $m = k$, truncated LPA makes its predictions by only considering nodes that have a walk of length k or smaller to the positive nodes. Note that λ plays a crucial role in determining the degree to which walks of increasing length contribute to the solution of LPA; for example, when λ is very small, the discriminant scores are essentially computed based on a very local neighborhood around the genes.

Figure 5.2 shows the performance of truncated LPA on GI and PI networks at several settings of λ. This figure reveals several important observations. First, by only considering first and second neighbors, we can achieve precision and AUROC similar to that of the exact solution of LPA. Furthermore, with an incorrect setting of the parameter λ, the performance of LPA could even decrease with increasing m. Third, with $\lambda \leq 0.5$ the truncated LPA converges to the exact solution very rapidly. Finally, the performance

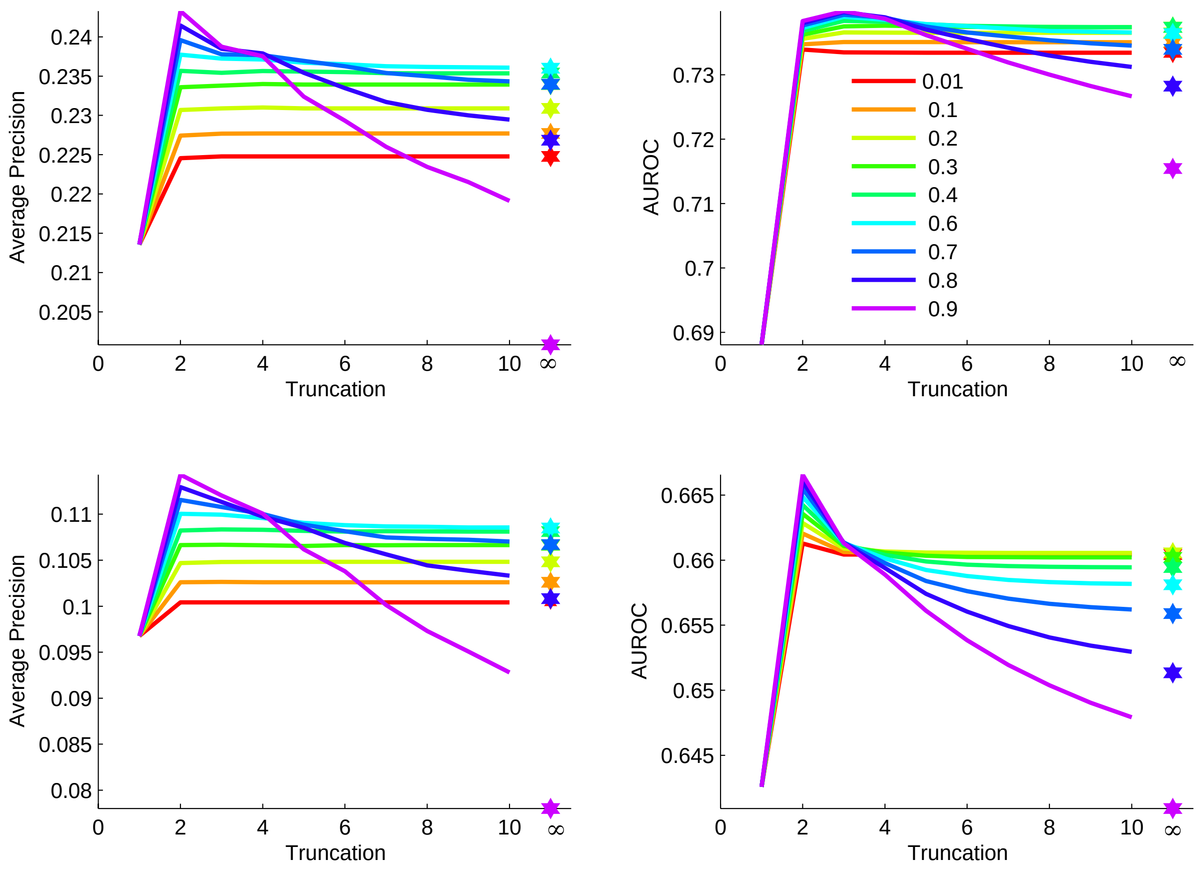


Figure 5.2: The performance of truncated LPA with truncation level $m = \{1, 2, ..., 10\}$ on PI (top) and GI (bottom) networks. The performance is shown in terms of average precision (left) and AUROC (right). Each line shows the performance with increasing m for a particular setting of the parameter λ. For each m and λ, the performance of the exact solution to LPA is shown by a star.

on the GI network reveals a widely reported property of genetic interactions: a gene's second-order neighbor is highly informative about its function [93, 19]. Note that when λ is large, the influence of walks of increasing length decays very slowly, which again explains the sharp decline in performance for $m > 3$.

One explanation for this sharp decline in performance with increasing walk lengths is the convergence of random walks to stationary distributions. Substituting for S into

Equation (5.1) we obtain:

$$\begin{aligned} \mathbf{f} &= (1-\lambda)\sum_{r=0}^{\infty}(\lambda S)^r \mathbf{y} && (5.2) \\ &= (1-\lambda)D^{1/2}\sum_{r=0}^{\infty}(\lambda P)^r D^{-1/2}\mathbf{y} && (5.3) \end{aligned}$$

where the second equality is obtained by noting that $S = D^{1/2}PD^{-1/2}$ and $(D^{1/2}PD^{-1/2})^r = D^{1/2}P^rD^{-1/2}$. Thus, the solution to LPA depends on increasing powers of the matrix P. It is well known that for a connected and non-periodic (or equivalently *ergodic*) Markov transition matrix (also known as a singly stochastic matrix) P, we have:

$$\lim_{r\to\infty} P^r = \mathbf{1}\pi^{\mathsf{T}} \qquad (5.4)$$

where π and $\mathbf{1}$ are vectors of dimension $n \times 1$. π is known as the *stationary distribution* [16]. In other words, for large r, P^r converges to a matrix with identical rows π^{T}. Thus, we note that if λ does not decay fast enough, adding multiple copies of P^r may decrease the performance. Below, we describe how to measure the convergence of P^r.

Note that for any positive vector $\mathbf{v}$ with $\sum_i v_i = 1$ we have that $\lim_{r\to\infty}\mathbf{v}^{\mathsf{T}}P^r = \pi^{\mathsf{T}}$. The *mixing time* of a random walk quantifies how large r should be for $\mathbf{v}^{\mathsf{T}}P^r$ to approximately converge to π^{T}. Formally, mixing time is defined as the smallest r such that:

$$||\mathbf{v}^{\mathsf{T}}P^r - \pi^{\mathsf{T}}|| < \epsilon \qquad (5.5)$$

where $||.||$ denotes the chosen measure of the distance [16]. A standard distance metric for measuring convergence is the *total variation distance* defined as half of the ℓ_1-norm: $TV = \frac{1}{2}\sum_i \left|[\mathbf{v}^{\mathsf{T}}P^r]_i - \pi_i\right|$ and by convention $\epsilon = \frac{1}{4}$. Thus, we define the mixing time of the Markov transition matrix P as the smallest r such that the total variation distance between $\mathbf{v}^{\mathsf{T}} P^r$, for a randomly chosen vector $\mathbf{v}$ with $\sum_i v_i = 1$, and π is less than $\frac{1}{4}$:

$$\frac{1}{2}(\sum_i \left|[\mathbf{v}^{\mathsf{T}} P^r]_i - \pi_i\right|) < \frac{1}{4}. \tag{5.6}$$

If P^r converges to $\mathbf{1}\pi^{\mathsf{T}}$ then when computing (5.3), with each P^t for t bigger than mixing time, we add a scalar proportional to $\sum_j \frac{d_{ii}}{d_{jj}}$ to each node i. Figure 5.3 shows the mixing time for the convergence of the PI and GI networks. As shown, the GI network has a mixing time of 3 (using a threshold of 0.25) which explains the sharp decline shown in Figure 5.2. Convergence for the PI network is slightly slower, which again is consistent with the results in Figure 5.2.

A related explanation for the fast convergence of the precision and AUROC of truncated LPA to the exact solution is the small average shortest distance in "small-world" networks, which scales with $\log \log n$ [67]. Small-world networks have a small diameter meaning that most nodes can be reached from every other by a small number of hops or steps. This property is attributed to the existence of hubs, which connect different communities (or clusters); once a node reaches a hub in the network, it can reach most other nodes through that hub [6].

To summarize, our results on the analysis and performance of truncated LPA allow us to make two conclusions: 1) due to the multiplication of P^r with λ^r, LPA assumes that the influence of walks of increasing length decreases geometrically. However, this form of decay may not be always appropriate; for instance, in the GI network walks of length two are more informative of co-functionality than those of length one. 2) Due to the small-world properties of empirical networks and fast convergence of P^r to a stationary distribution, we can effectively use truncated LPA instead of solving LPA exactly. In the next section, we use these two observations to propose a modified version of LPA that is faster and more accurate.

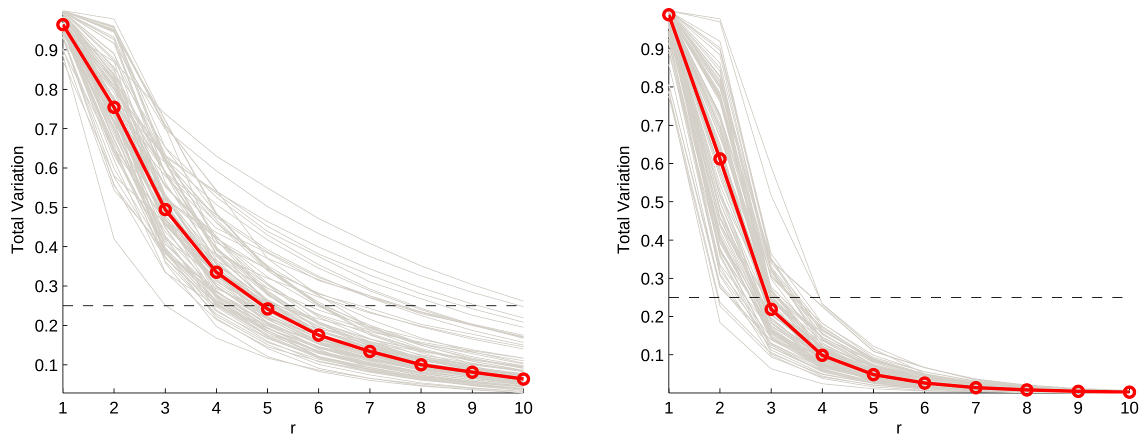


Figure 5.3: Convergence of random walks according to the total variation distance. To generate each line, we start from a random vector $\mathbf{v}$, with $\sum_i v_i = 1$. For each value of $r = \{1, 2, ..., 10\}$, we measure the convergence of $\mathbf{v}^\mathsf{T} P^r$ to the stationary distribution π using the total variation distance (i.e. the y-axis shows $\frac{1}{2}|\mathbf{v}^\mathsf{T} P^r - \pi|_1$ as a function of r, where $|a|_1$ is the ℓ_1-norm of a). The grey lines show the total variation distance for 100 randomly chosen vectors $\mathbf{v}$ with increasing r. The figure on the left shows the convergence on the PI network and the figure on the right shows the convergence on the GI network. The red lines show the median of total variation distances on each network.

5.3.2 Weighted Proximal Propagation

In this section we propose *Weighted Proximal Propagation (WPA)*, our modified version of LPA that 1) doesn't assume a geometric decay in down-weighting walks of increasing length and 2) is faster than LPA by only using local (short) walks. In particular, we define WPA as:

$$\mathbf{f} = \sum_{r=0}^{m} \beta_r (S)^r \mathbf{y} \tag{5.7}$$

where $\beta = [\beta_1, ..., \beta_m]^\mathsf{T}$ consists of the model parameters. The difference between (5.7) and truncated LPA is that here the β_i's are free to be optimized separately, for walks of different lengths. WPA allows us to set the truncation level m and set the parameters β. A brute force approach to determine both is to perform a grid search on the training data, which is rather costly. We set $m = 3$ as we have found this setting to work well on a variety of networks.

Recall that the $(i,j)^{\text{th}}$ entry of S^m, i.e. $[S^m]_{ij}$, is non-zero only if there is a walk of length m between nodes i and j. Note that the existence of a walk of length m does not exclude the existence of a shorter walk—for example, node i can be connected to node j with both a walk of length one and two. As a result, the entries of S, S^2, and S^3 may overlap—such overlaps help to discern community structure as two nodes in a tightly knit community share more paths of short length compared to pairs of nodes that are not in the same community [66] (for example, as shown for the modular network in Figure 2.2).

We propose to obtain values of β using Linear Discriminant Analysis (LDA) [9]. Intuitively, LDA estimates a linear combination of walks of length 1 to m using the coefficients β; the coefficients are set to maximize the difference between the scores of the positives and the non-positives. Denote the input to node i from a walk of length r by $x_i^{(r)} = \sum_j [S^r]_{ij} y_j$. Thus, the final score of node i is computed as $f_i = \sum_r \beta_r x_i^{(r)}$. Using the linear regression interpretation of LDA [9] , we determine β by minimizing the squared error for each f_i with respect to $\hat{y}_i$:

$$\underset{\beta}{\operatorname{argmin}} \sum_i (\sum_r \beta_r x_i^{(r)} - \hat{y}_i)^2 \tag{5.8}$$

where $\hat{y}_i$ is set to $\frac{1}{n^+}$ for positive examples and $-\frac{1}{n^-}$ for non-positives. The solution to (5.8) is given by $\beta = (X^\mathsf{T} X)^{-1} X^\mathsf{T} \hat{\mathbf{y}}$; where X is an $n \times r$ matrix whose i^{th} row is given by $[x_i^{(1)} \ldots\ x_i^{(r)}]$. Prior to solving for β, we center the columns of X to have mean zero. Note that in (5.8) we are using $\mathbf{y}$ to estimate x_i's, so to avoid overfitting, we perform LDA in cross-validation[1].

Although our choice of setting $m = 3$ is motivated based on empirical observations, previous studies have reported similar results on a variety of other networks [13, 14]. In

[1]We leave out a random portion of positives, assume they were non-positives, obtaining $\mathbf{y}_{cv}$, we use $\mathbf{y}_{cv}$ to compute X

the next section, we show that $m = 3$ holds for several other networks as well. However, since we set the coefficients of WPA using LDA-based linear regression, as future work, we can also determine m by performing forward-stage-wise search where we determine m based on a penalized likelihood score (such as Akaike Information Criterion (AIC) or Bayesian Information Criterion (BIC) [9]).

Unlike with the label propagation algorithm, the coefficients β are not constrained to be positive or smaller than one. This flexibility is required to accurately predict node labels in the presence of arbitrary patterns of label distribution. For instance, Figure 5.4 shows an example scenario using a bipartite network. Similar to the intuition about *negative* genetic interactions[2], our assumption is that nodes connected by a walk of length two have the same function. In this case, label propagation's geometric decay assumption leads to erroneous support for first-degree neighbors, whereas WPA correctly identifies second-degree neighbors as the main targets while exploiting the "negative information" presented by the first-degree neighbors.

To summarize, our major observation is that labeling patterns of nodes can vary between different networks. Therefore, we can improve on the performance of LPA by *learning* the contribution of walks of increasing length for a given network. Furthermore, we have shown that the performance of truncated LPA equals or exceeds that of the exact solution to LPA. This observation explains the success of local neighborhood-based heuristics [65, 89]. In WPA we exploit these properties by setting the truncation level to $m = 3$, in addition to fitting the parameter β to allow separate control over the contribution of walks of a given length to the discriminant scores.

[2]Recent studies have shown that second-order neighbors in negative genetic interaction networks are more likely to share the same function than first degree neighbors [93, 19].

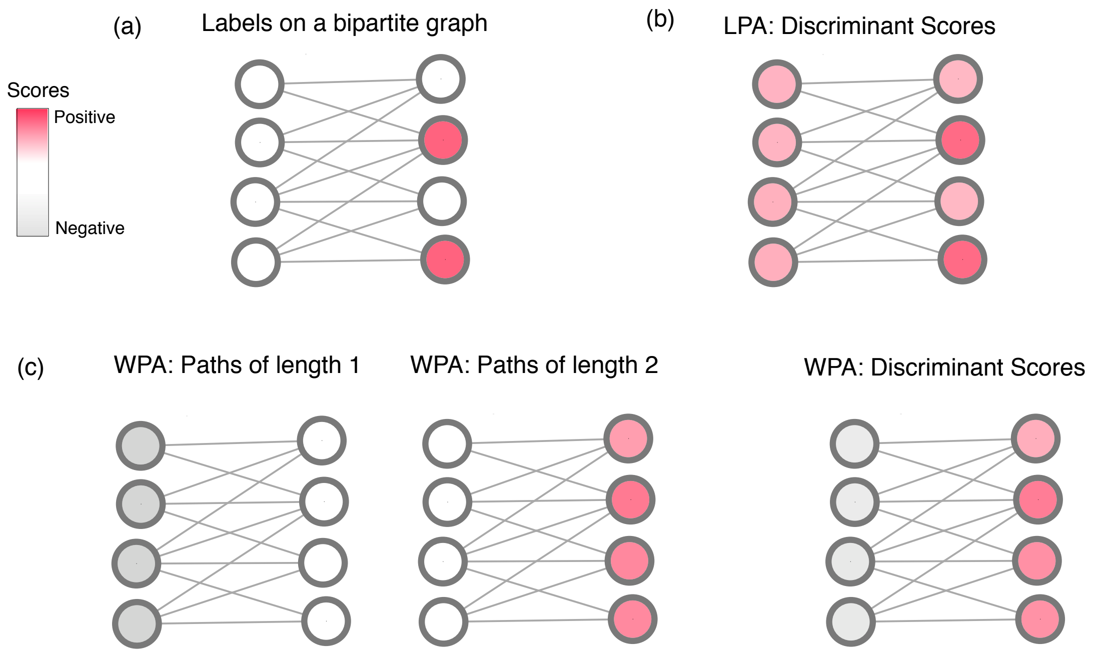


Figure 5.4: An example bipartite network where the nodes are divided into two groups that link only to nodes in the opposite group. This example is meant to resemble a small genetic interaction network where genes connected with a walk of length 2 are more likely to be co-complexed and thus perform the same function. In this scenario, LPA fails to up-weight the second-order neighbors. In contrast, WPA automatically assigns a high coefficient to walks of length 2 and thus the final discriminant score reflects the initial pattern that nodes in the same cluster have the same label.

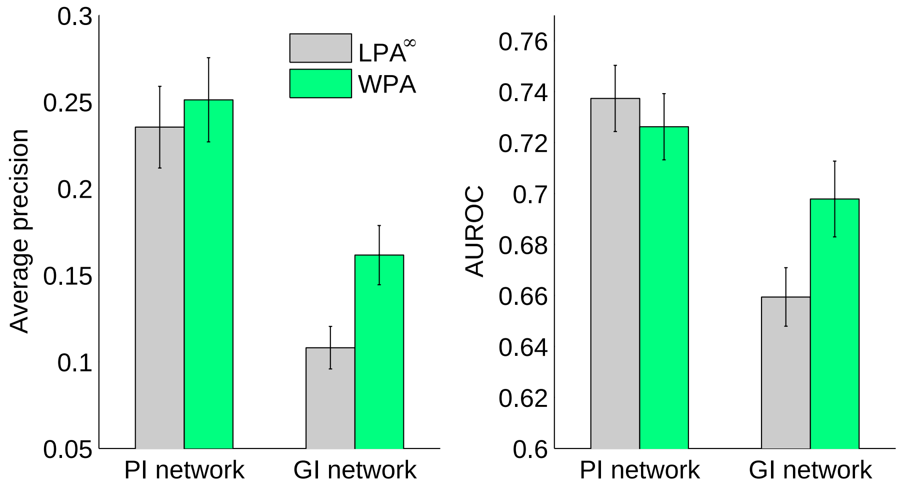


Figure 5.5: The performance of exact LPA and WPA (using walks of length up to three) on PI and GI networks. The performance is shown in terms of both AUROC (left) and average precision (right). Error bars show the standard error.

5.4 Results

In this section, we compare the performance of WPA with exact LPA on PI and GI networks. As well, we apply WPA to three other problems: predicting gender from five Facebook networks, political view (conservative or liberal) from a blog web-links network, and predicting 381 patent categories from a patent-citation network.

5.4.1 Predicting Gene Function with WPA

Figure 5.5 shows the performance of exact LPA and WPA. Using only walks of length up to three, WPA performs similarly to (PI network) or significantly better than (GI network) LPA. Since the influence decay model of LPA holds for PI networks, WPA performs similarly to LPA on this network. On the other hand, WPA is able to exploit the significance of second-order interactions in GI networks and thus improves on the performance of LPA in this setting. In Figure 5.5, the performance is shown for predicting 47 non-redundant GO terms (obtained from [62]) which have at least 30 annotations. The mean and median number of annotations in these 47 categories are 110.7 and 96,

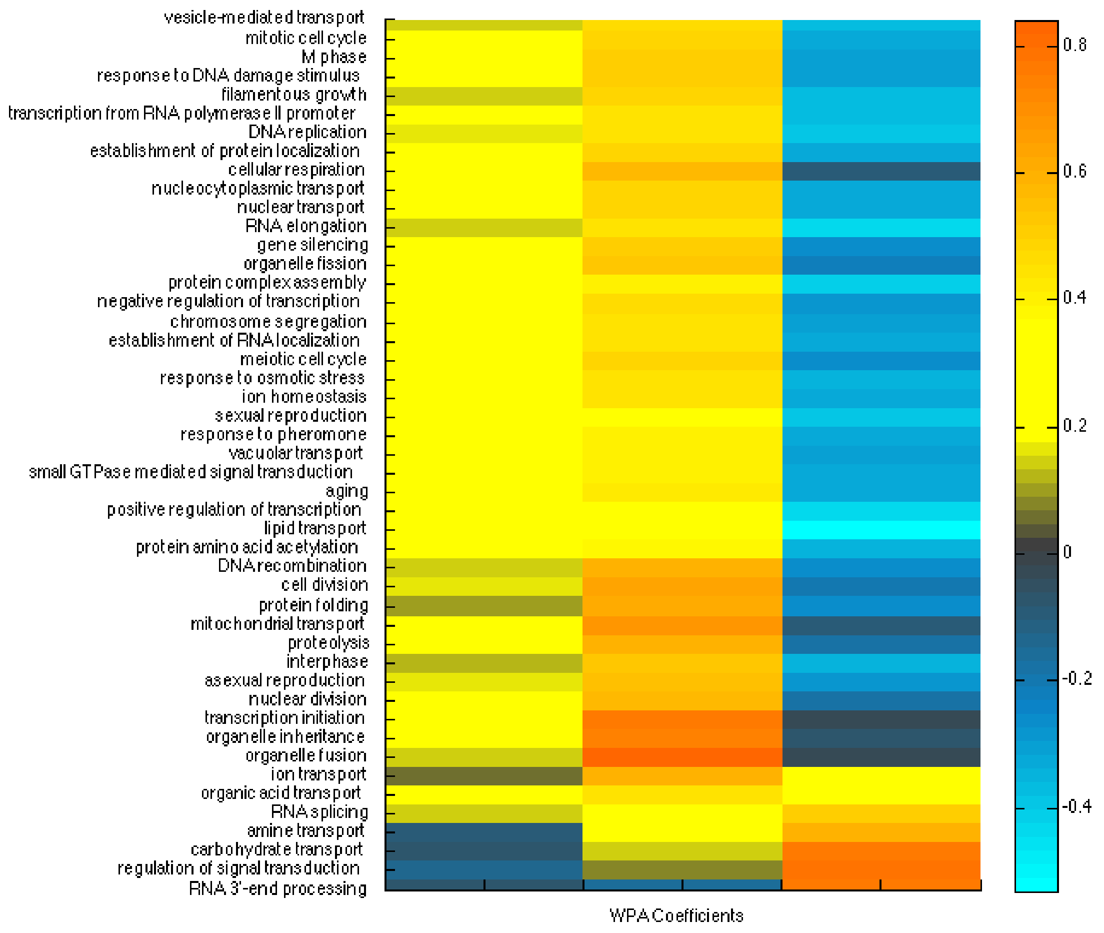


Figure 5.6: The coefficients assigned to walks of length one to three for the PI network when predicting 47 GO categories. The columns show the magnitude of the coefficients in order (column 1 represents β_1 etc.). The rows are ordered by clustering the coefficients. The coefficients β_r's are scaled to have an absolute sum of 1.

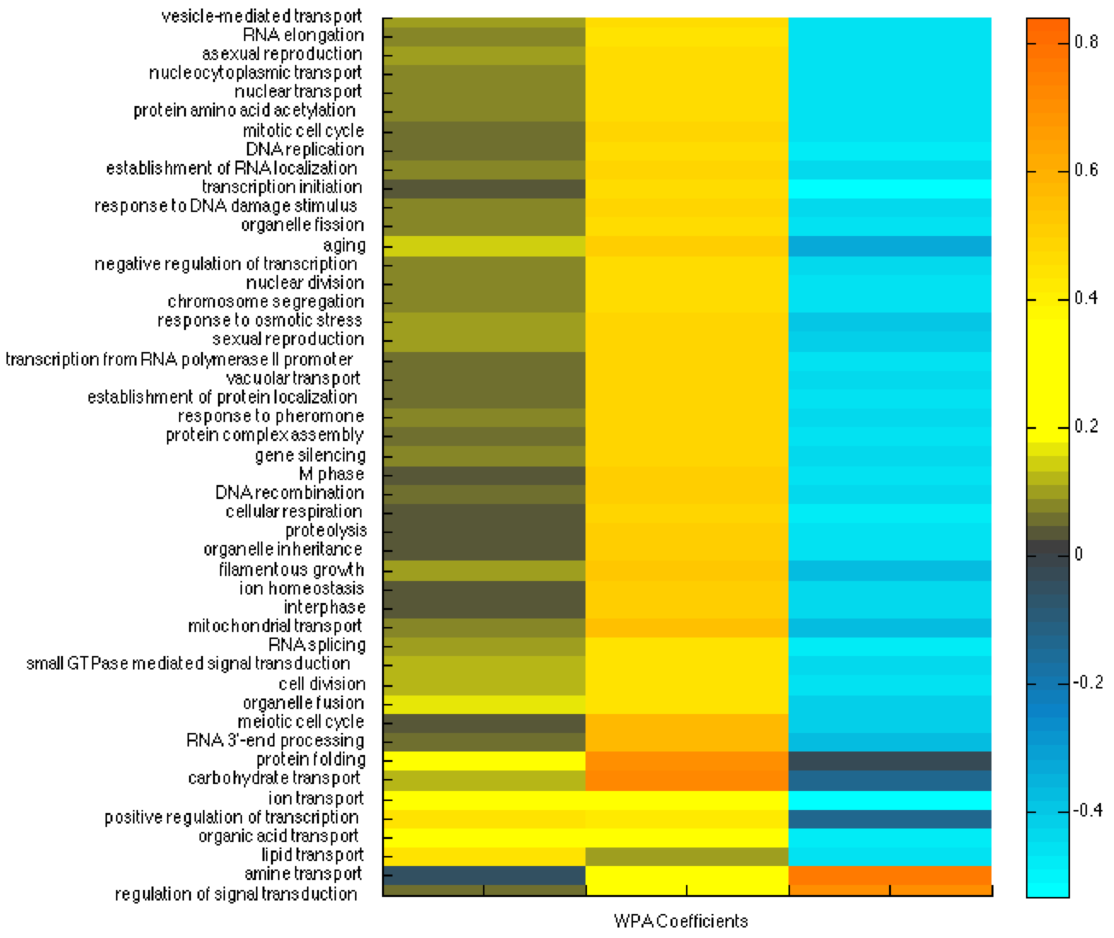


Figure 5.7: The coefficients assigned to walks of length one to three for the GI network when predicting 47 GO categories. The columns show the magnitude of the coefficients in order (column 1 represents β_1 etc.). The rows are ordered by clustering the coefficients. The coefficients β_r's are scaled to have an absolute sum of 1.

respectively. In general, we didn't observe significant differences in the performance based on number of annotations.

Figures 5.6 and 5.7 show the assigned coefficients for the 47 GO categories. With a few exceptions, in both networks, walks of length two have a larger coefficient than those of length one. This observation is more exaggerated for the GI network. Furthermore, walks of length three are often assigned a negative coefficient. There are two explanations for this observation. One explanation for this phenomenon may be the high clustering coefficient (CC) of the small-world network. Intuitively, CC is a measure a "cliqueness" or how often a node's first-degree neighbors are also second-degree neighbors. Due a high CC, walks of length two will be given a higher coefficient as they also reflect direct neighbor connections. Another explanation is that since $\beta_3 < 0$, the probability distribution of walks of length three from a node i to the other nodes is approaching the equilibrium distribution and so WPA "subtracts off" the equilibrium distribution component from walks of length two by assigning a negative coefficient to walks of length three.

One interesting observation from Figures 5.6 and 5.7 is that there are two clusters of GO categories; the majority of the functions are in one big cluster where $\beta_2 > \beta_1$, and a few "outliers" fall in the second category where β_2 is comparatively smaller. In particular, for both networks "regulation of signal transduction" and "amine transport" fall under the second cluster where β_2 is small, β_3 is positive, and β_1 is negative. This observation can be explained by the fact that for both "regulation of signal transduction" and "amine transport" functions, only a small fraction of positive genes have a direct interaction (i.e. the positives are sparsely connected to each other), which result in a low β_1. Furthermore, performance of LPA and WPA in terms of average precision is much lower on these categories compared to the average performance: for example, for predicting "regulation of signal transduction" with the GI network, WPA results in average precision of 0.02 and LPA results in average precision of 0.03, whereas the overall average precision of WPA is 0.15. In general, the outlier functions tend to be the functions

Table 5.1: Performance of LPA and WPA on five Facebook networks (from five universities), the Blogs network, and the Patent network.

	AUROC		Precision	
	WPA	LPA	WPA	LPA
Caltech	**0.726**	0.593	**0.662**	0.531
Georgetown	**0.768**	0.588	**0.703**	0.498
Oklahoma	**0.845**	0.631	**0.807**	0.581
Princeton	**0.786**	0.644	**0.727**	0.553
UNC	**0.817**	0.625	**0.772**	0.552
Blogs	**0.954**	0.930	**0.961**	0.931
Patent	0.981	**0.992**	**0.642**	0.610

where the positives are not well connected and so local neighborhood connections are not very predictive of shared functionality.

5.4.2 Other Applications

In this section, we investigate the performance of WPA and LPA on predicting gender from five Facebook networks, political view from a blog web-link ("Blogs") network, and 381 patent categories from a patent-citation ("Patent") network.

Table 5.1 summarizes the results for these networks. As shown, in most cases WPA considerably improves the performance of LPA despite only requiring three matrix-vector products. By examining the coefficients found by WPA, we can gain insight into the characteristic patterns in how labels are distributed within the network structure. For example, we found that a person is likely to have the same gender as her (or his) friends-of-friends ($0 < \beta_1 < \beta_2$). In addition, the coefficient assigned to walks of length three is negative ($\beta_3 < 0$); in the absence of "negative" information from friends-of-friends-of-friends, the performance of label prediction drops to the baseline (almost random performance) and so this information, in combination with gender of friends-of-friends, is crucial for accurately predicting gender. One explanation for this phenomenon can be the convergence of random walks to the equilibrium distribution. By assigning a negative coefficient to walks of length three, WPA is able to "subtract off" the contribution from

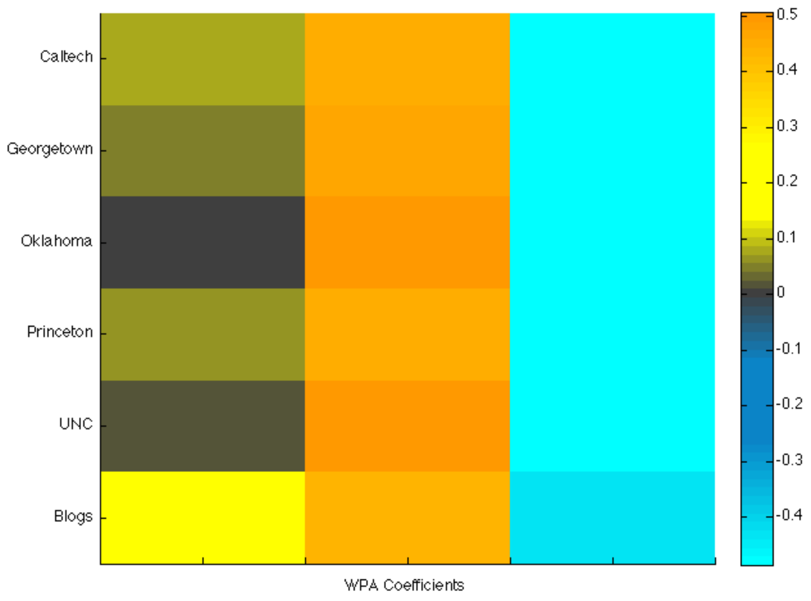
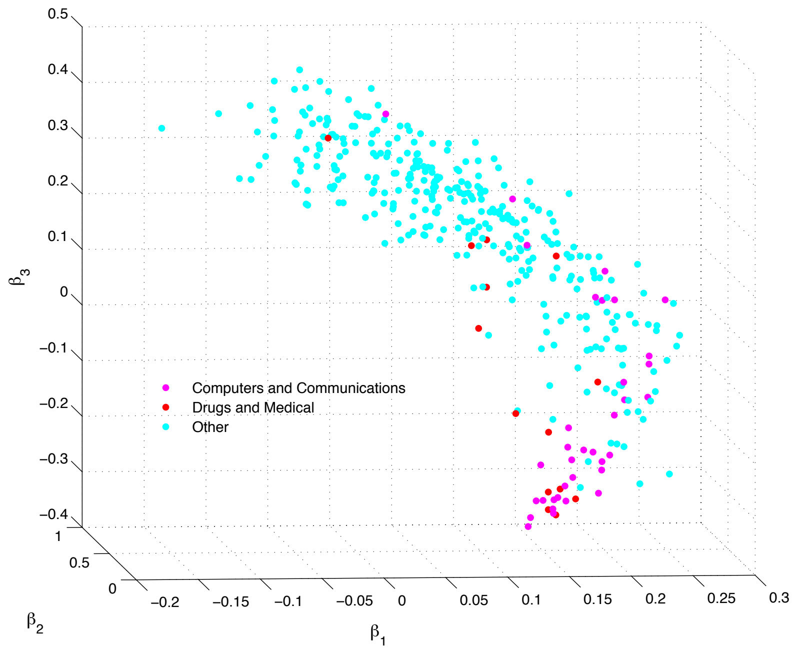


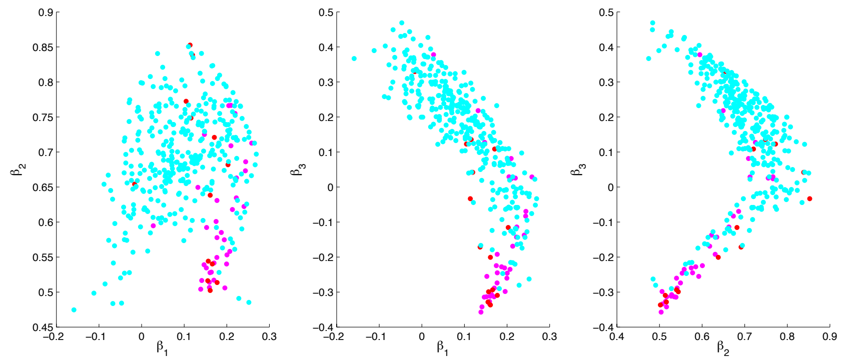
Figure 5.8: The coefficients of WPA assigned to five Facebook networks and the Blogs network. As shown, walks of length two are most predictive of gender (in Facebook) and political view (in Blogs), resulting in larger values of β_2.

the equilibrium distribution and thus improve on the performance of LPA. Similar patterns hold for the Blogs networks; most blogs with the same political view are connected to each other by walks of length no more than two. Accordingly, one explanation for this phenomenon is the existence of "liberal" and "conservative" hubs which connect blogs with the same view.

Figure 5.9 shows the coefficients β assigned by WPA when predicting 381 patent categories. These categories were also classified into six technology types (Chemical, Computers and Communications (CC), Drugs and Medical (DM), Electrical and Electronics (EE), Mechanical, and Others) [32]; without using any information about the technology types, the assigned coefficients are more similar for patent categories of the "newer" technology types (CC and DM) compared to the "older" ones (Chemical, EE, Mechanical and Others). Note that there are two distinct trends: patents whose direct neighbors are assigned a negative coefficient and whose third degree neighbors are as-



(a)



(b)

Figure 5.9: The coefficients of WPA assigned to the Patent network when predicting 381 patent categories. Each dot corresponds to one patent category. These 381 patent categories can be categorized into six technology types. We further categorize these six technology types into two groups made up of "newer" patents (Computers and Communications, Drugs and Medical) and one group made up of the "older" patents ("Other" which is made up of Textiles and Agriculture, Mechanical, Chemical, and Electric and Electronics). The coefficients assigned by WPA are similar for patent categories that are in the same technology type. Plot (a) shows the coefficients $(\alpha_1, \alpha_2, \alpha_3)$ in three dimensions, and plot (b) shows all pairwise combinations of coefficients, with (α_1, α_2) on the left, (α_1, α_3) in the middle, and (α_2, α_3) on the right.

signed a large positive coefficient, versus patents whose direct neighbors are assigned a positive coefficient and whose third degree neighbors are assigned negative coefficients. The former trend mostly comprises Mechanical or Other types of patents (which are more likely to be older) and the latter mostly comprises Computers or Communications and Drugs and Medical patents.

5.5 Summary

In this chapter, we have introduced WPA, a faster alternative to LPA for predicting binary node labels from networks. As we have shown, WPA also equals or exceeds LPA in accuracy and precision on a variety of networks. The improvement is most drastic where the labeling pattern of nodes does not fall under the *influence decay* model of LPA that down-weights the propagation received from walks of increasing length, according to a fixed exponential scale. Another advantage of WPA is that its solution only requires a few matrix-vector products and thus its complexity is $O(m)$ where m is the number of non-zero edges in the network. This allows us to apply WPA to very large networks, such as the patent network with its 3M nodes and 16M edges.

Chapter 6

Conclusions and Future Work

The availability of a large number of genomics and proteomics datasets presents an opportunity for predicting the function of uncharacterized genes. However, existing approaches either do not scale to large genomes, or gain scalability by making assumptions that degrade their accuracy. In this dissertation, we design and develop algorithms that scale to large datasets, and can be used to accurately predict gene function from arbitrary datasets and query genes, on demand. By combining scalability and accuracy, we can now take advantage of the persistent improvement and proliferation of genomics and proteomics datasets to make up-to-date predictions for large genomes such as the human genome, rather than having to resort to static databases that attempt to pre-compute a large number of pre-defined queries.

In Chapter 2, we reviewed the body of research on predicting gene function from heterogeneous data sources. As discussed, a common approach for combining heterogeneous data is to first represent each dataset as a network. In this context, we have described existing approaches for combining multiple networks and making predictions from the combined network. We showed that the existing approaches fall under one of two categories: those that make weak assumptions but solve challenging optimization problems and therefore do not scale to larger genomes or less sparse networks; or those that gain scalability by making stronger limiting assumptions that greatly decrease the accuracy of their predictions.

In Chapter 3, we presented the GeneMANIA framework for accurate and efficient prediction of gene function from multiple networks. We achieve scalability by formulating a constrained linear regression problem for constructing query-specific network combinations, and by using a label propagation algorithm for making predictions from the combined network. By exploiting a large amount of high-throughput data while accounting for redundant and irrelevant networks, we automatically reduce or remove false positive interactions, and thus improve on the accuracy of existing methods.

In Chapter 4, we described extensions to the label propagation algorithm, HLProp and HLBias, for incorporating the ontological organization of gene functions when making predictions. We showed that this type of prior information can significantly improve the performance of regular label propagation. One observation was that although HLBias merely improves the performance in terms of AUROC, HLProp improves the performance in terms of both AUROC and average precision. As well, we showed that a simple approximation to HLProp results in competitive performance while having a much lower computational cost.

In Chapter 5, we further considered the general problem of predicting binary node labels from arbitrary networks. There, we closely studied the empirical behavior of label propagation on real-world networks. We explained the reasons for the successes of some local neighborhood-based methods in terms of their relation to label propagation. Motivated by our observations, we introduced a new label propagation algorithm, WPA, that exploits the empirical properties of real-world networks to make faster and more accurate predictions. WPA achieves its accuracy by *independently weighting* the contribution of random walks of increasing length to support a wider range of labeling patterns. We showed that WPA outperforms label propagation on networks from varying domains. WPA is scalable to very large networks as its solution only requires a few matrix-vector products.

One practical observation from our experiments in Chapter 3, 4, and 5 is on the selection of negative examples. In Chapter 3 and 4, we observed that explicitly assigning negative examples based on sibling categories may not always improve the performance, however, in Chapter 4, we showed that we can considerably improve the predictive performance by setting an initial label bias according to the ontology structure (e.g. by using the Down Propagation method). This result shows that the ontology structure embodies information that is useful for improving function prediction and negative selection. However, this information may not always be available (for example when GeneMANIA is used as an online prediction server). In Chapter 5, we assigned all non-positive genes the same initial label (equal to zero). In experimentation, we have observed that by simply using binary labels (as done in Chapter 5), we obtain similar performance as when assigning "average label bias" to the unlabeled genes (as done in Chapter 3). The results from Chapter 3 and 5 our complementary in the sense that in Chapter 3, the initial label bias assigned to unlabeled genes tends to be close to the label of the negative examples—this is because in gene function prediction, the number of potential negative examples is orders of magnitude larger than the number of positive examples. Thus, our results suggest that in the absence of knowledge about ontology structure, we can obtain good performance by simply setting the initial labels of all non-positives to zero.

In a broader context, the approaches that we have introduced in this thesis are promising for solving any prediction problem for which there is a large amount of unlabeled data available that can be used to associate the examples (instances) in various ways. For instance, a relevant problem is to predict diagnosis or prognosis for a patient based on multiple similarity networks constructed over the patients. These similarity networks can be derived from different types of molecular data that are available for a group of people such as their gene-expression profiles, SNPs, and epigenomics features. In other applications, our algorithms can be used to predict a person's preference for a product based on multiple networks, for example, those derived from friendship ties and demographic information. Our extensions of label propagation for incorporating ontology structure may also be useful in predicting genes that are involved in different diseases. This is motivated by recent studies that have constructed networks of disease-disease similarities (e.g. the *Diseaseome* network [30]).

In summary, one recurring observation is that for exploiting the large scale datasets considered in this thesis, well-formulated linear models have resulted in predictive performance that equals or exceeds that of more complex models. Furthermore, obtaining the solution to a variety of models requires solving sparse systems of linear equations. We have shown that such systems of equations that arise from appropriate models of data can be solved efficiently, often requiring a few matrix-vector products. These observations are likely to hold in other "data-rich" domains.

Future Work

Here we suggest directions for future research that are motivated by the present project.

The probabilistic interpretation of label propagation motivates several directions for future work. Accordingly, an initial label bias can be interpreted as a prior probability with a fixed precision. In Chapter 4, we showed that setting prior probabilities for labels according to the ontological structure of gene functions (Section 4.2.1), or the proportion of observed positives for a given prediction task (Section 3.3.2), can greatly improve the predictive performance. An extension to this work is to learn the prior precisions from data. For example, one can use the GO evidence codes to estimate prior precisions. Another extension would be to set the prior precisions by using profile-based data. For example, as opposed to using domain composition data in the form of a shared domain network, we can use the domain data to learn the prior precisions. This extension can further speed up label propagation on large-scale datasets—networks that are constructed from feature-based data are often dense, whereas solving for the solution to label propagation scales with the number of non-zero edges in the networks.

As we showed in Section 2.6.2, we can use the probabilistic interpretation of LPA to compute the posterior variance for each discriminant score. One area of future work would be to compute these posterior variances, combine them with the discriminant scores, and assess the performance. A related topic is to investigate how to convert the discriminant scores to confidence scores. In particular, the discriminant scores returned by label propagation depend on the number of positive examples and so discriminant scores of a gene for two different classification tasks are not directly comparable. A simple method for computing confidence scores is to normalize the initial label vector (e.g. to have a length of one) and to fit a sigmoid function to the discriminant scores (e.g. as done in [78]). Another possibility is to perform cross-validation to adjust the discriminant scores based on the discriminant scores assigned to the left out positive examples.

In Chapter 5, we showed that we can define three different versions of label propagation by changing how we normalize the Laplacian matrix of an affinity network. This normalization has a large impact on the accuracy of the obtained solution. In particular, the symmetric normalization up-weights the discriminant scores of nodes with larger weighted degrees, which works well for the purpose of gene function prediction—a recent study showed that because of multi-functionality of hubs, node degrees can be highly predictive of gene function [29]. A future direction is to further investigate the effect and potential of different normalization methods and their impact on node ranking. Another area for future work is to further investigate the observations reported in [29] by removing high degree nodes and then assessing the predictive performance. Removing hubs that connect different types of functional modules (e.g. "date hubs" [33]) may result in improved performance.

As we showed in Chapter 5, the iterative solution of label propagation converges very quickly on real-world networks. These observations will be very useful when designing algorithms for combining and making predictions from networks. One obvious extension to WPA is to learn the random walks parameters for several networks and thus allow for predictions from multiple networks simultaneously. Another possible extension of WPA is to learn *separate weight* for random walks originating from each node, as opposed to having one parameter for all random walks of a given length for all nodes.

In conclusion, we have introduced accurate and scalable algorithms for predicting gene function form multiple networks. As we have discussed, our observations and algorithms are instructive for solving other prediction problems for which there is a large amount of unlabeled data available that can be used to associate the examples (instances).

Appendix A

In this appendix, we summarize the data sources we used to construct functional linkage networks. To construct networks from both profile-based data and network-based data we use the Pearson Correlation Coefficient (PCC) to measure pairwise similarities. We set all negative PCC values to zero. The PCC networks are often dense and there is often a large proportion of non-zero edges. This is because many pairs of genes may have a very small yet non-zero correlation coefficient. Since the efficiency of algorithms for predicting node labels from networks depend on the number of non-zero edges, we sparsify dense PCC networks. To do so, for each gene we keep its top k nearest neighbor neighbors (NN) and set the rest to zero. Subsequent to sparsifying a network, we ensure that is symmetric by setting w_{ij} to the maximum of w_{ij} or w_{ji}. Empirically, we have observed that we can set $k = 100$ without sacrificing performance [60]. Tables 6.1-6.6 provide references for all datasets that we used to construct networks.

Table 6.1: Data sources used for constructing eight networks in Human. Most of these data was collected from the HPRD database [57] (downloaded November 2007), otherwise we provide the PubMed ID (PMID) of the corresponding publication. We construct networks from each dataset by using the PCC to measure pairwise similarities between genes.

Source	Type
HPRD	Shared domain composition
HPRD	Co-complexed
HPRD	Shared regulation
HPRD	Shared prost-transcriptional modification
HPRD	Co-expression
HPRD	Protein interaction
HPRD	Shared phenotype (OMIM disease)
PMID:15075390	Co-expression

Table 6.2: Data sources used for constructing 15 networks in Yeast (Yeast15). We construct networks from each dataset by using the PCC to measure pairwise similarities between genes. PMID indicates the corresponding PubMed ID for each publication.

Source	Publication	Type
PMID:9843569	Spellman, Mol Biol Cell, 1998	Co-expression
PMID:10929718	Hughes, Cell, 2000	Co-expression
PMID:1110252	Gasch, Mol Biol Cell, 2000	Co-expression
PMID:16880382	Chua (DM), PNAS, 2006	Co-expression
PMID:16880382	Chua (OE), PNAS, 2006	Co-expression
PMID:10657304	Roberts, Science, 2000	Co-expression
PMID:12897782	Yvert, Nat Genet, 2003	Co-expression
PfamA	2006	Shared domain composition
PfamB	2006	Shared domain composition
PMID:14718668	Giaever, PNAS, 2004	Shared phenotype
PMID:12399584	Lee, Science, 2002	Shared regulation
PMID: 15343339	Harbison, Nature, 2004	Shared regulation
PMID:16429126	Gavin, Nature, 2006	Protein interaction
PMID:16554755	Krogan, Nature, 2006	Protein interaction
PMID:14562095	Huh, Nature, 2003	Co-localization

Table 6.3: Data sources used for constructing seven networks in E.coli. This datasets were collected by Hu and colleagues [39]. We construct networks from each dataset by using the PCC to measure pairwise similarities between genes. PMID indicates the corresponding PubMed ID for each publication.

Source	Publication	Type
PMID:19402753	Hu. Plos Biol, 2009	Protein interaction
M3D database	M3D database	Co-expression
PMID:19402753	Hu, Plos Biol, 2009	Shared Operons
PMID:19402753	Hu, Plos Biol, 2009	Gene fusion
PMID:19402753	Hu, Plos Biol, 2009	Co-inheritance (shared phylogenetic profile)
PMID:19402753	Hu, Plos Biol, 2009	Close distance in chromosome features 1
PMID:19402753	Hu, Plos Biol, 2009	Close distance in chromosome features 2

Table 6.4: Data sources used for constructing ten mouse networks. This datasets were collected by Pena-Castillo and colleagues [76] and used in the MouseFunc challenge. We construct networks from each dataset by using the PCC to measure pairwise similarities between genes. PMID indicates the corresponding PubMed ID for each publication.

Source	Publication	Type
PMID:1558831	Zhange, J. Biol, 2004	Co-expression
PMID:15075390	Su, PNAS, 2004	Co-expression
SAGE Libraries	2007	Co-expression
OPHID	2007	Protein interaction
Pfam	2007	Shared domain composition
InterPro	2007	Shared domain composition
bioMART	2007	Co-inheritance (shared phylogenetic profile)
Inparanoid	2007	Co-inheritance (shared phylogenetic profile)
MGI	2007	Shared phenotype
OMIM	2007	Shared phenotype (disease)

Table 6.5: Data sources used for constructing 44 networks for yeast (Yeast44). For genetic interaction networks that reported both positive and genetic interaction scores, we construct four functional linkage networks: one direct negative genetic interaction network, one direct positive genetic interaction network, one PCC network constructed from the positive genetic interactions, and one PCC network constructed from the negative genetic interactions. For the protein interaction data, we use a direct interaction network as well as one constructed by using the PCC. PMID indicates the corresponding PubMed ID for each publication.

Source	Publication	Type	Number of networks
PMID:10657304	Roberts, Science 2000	Co-expression	1
PMID:10929718	Hughes, Cell 2000	Co-expression	1
PMID:11102521	Gasch, Mol Biol Cell. 2000	Co-expression	1
PMID:11805826	Gavin, Nature 2002	Protein interaction	2
PMID:11805837	Ho, Nature 2002	Protein interaction	2
PMID:14562095	Huh, Nature 2003	Co-localization	1
PMID:14718668	Giaever, PNAS, 2004	Co-regulation	1
PMID:14764870	Tong, Science 2004.	Genetic interaction	2
PMID:16093310	Miller, PNAS 2005	Protein interaction	2
PMID:16269340	Schuldiner, Cell 2005 (PS)	Genetic interaction	4
PMID:16319894	Ptacek, Sciene 2005	Protein interaction	2
PMID:16429126	Gavin, Nature 2006	Protein interaction	2
PMID:16487579	Pan, Cell 2006 (SL)	Genetic interaction	4
PMID:16554755	Krogan, Nature 2006	Protein interaction	2
PMID:16880382	Chua, PNAS 2006	Co-expression	2
PMID:17200106	Collins, Mol Cell Proteomics 2007	Protein interaction	2
PMID:17314980	Collins, Nature 2007 (PS)	Genetic interaction	4
PMID:17923092	McClellan, Cell 2007	Genetic interaction	2
PMID:18467557	Tarassov, Sciene 2008	Protein interaction	2
PMID:18676811	Lin, Gens Dev 2008	Genetic interaction	2
PMID:18719252	Yu, Science 2008	Protein interaction	2
PMID:9843569	Spellman, Mol Biol Cell. 1998	Co-expression	1

Table 6.6: Data sources used for constructing 38 networks for fly. We construct two networks from each protein interaction dataset: one direct network and one PCC network. The gene expression datasets were downloaded from GEO, below we provide the corresponding GEO ID for the corresponding publications.

Accession	Source	Type	Number of networks
GDS2674	Gene Expression Omnibus	Co-expresions	1
GDS2399	Gene Expression Omnibus	Co-expresions	1
GDS2485	Gene Expression Omnibus	Co-expresions	1
GDS516	Gene Expression Omnibus	Co-expresions	1
GDS1937	Gene Expression Omnibus	Co-expresions	1
GDS2675	Gene Expression Omnibus	Co-expresions	1
GDS1842	Gene Expression Omnibus	Co-expresions	1
GDS2504	Gene Expression Omnibus	Co-expresions	1
GDS2272	Gene Expression Omnibus	Co-expresions	1
GDS1526	Gene Expression Omnibus	Co-expresions	1
GDS23	Gene Expression Omnibus	Co-expresions	1
GDS444	Gene Expression Omnibus	Co-expresions	1
GDS732	Gene Expression Omnibus	Co-expresions	1
GDS443	Gene Expression Omnibus	Co-expresions	1
GDS667	Gene Expression Omnibus	Co-expresions	1
GDS664	Gene Expression Omnibus	Co-expresions	1
GDS653	Gene Expression Omnibus	Co-expresions	1
GDS1690	Gene Expression Omnibus	Co-expresions	1
GDS2665	Gene Expression Omnibus	Co-expresions	1
GDS2071	Gene Expression Omnibus	Co-expresions	1
GDS2479	Gene Expression Omnibus	Co-expresions	1
GDS1911	Gene Expression Omnibus	Co-expresions	1
GDS1739	Gene Expression Omnibus	Co-expresions	1
GDS2673	Gene Expression Omnibus	Co-expresions	1
GDS1977	Gene Expression Omnibus	Co-expresions	1
GDS1877	Gene Expression Omnibus	Co-expresions	1
GDS1686	Gene Expression Omnibus	Co-expresions	1
GDS2830	Gene Expression Omnibus	Co-expresions	1
GDS2784	Gene Expression Omnibus	Co-expresions	1
GDS2228	Gene Expression Omnibus	Co-expresions	1
GDS1395	Gene Expression Omnibus	Co-expresions	1
GDS602	Gene Expression Omnibus	Co-expresions	1
PMID14605208	BIOGRID	Protein interaction	2
PMID15575970	BIOGRID	Protein interaction	2
Interpro domains	Interpro	Shared domain composition	1
Pfam domains	Pfam	Shared domain composition	1

Bibliography

[1] L.A. Adamic and N. Glance. The political blogosphere and the 2004 U.S. election. *WWW-2005 Workshop on the Weblogging Ecosystem*, 2005.

[2] S. Aerts, D. Lambrechts, S. Maity, P. Van Loo, B. Coessens, and F. De Smet et al. Gene prioritization through genomic data fusion. *Nat Biotech*, 24:537–544, 2006.

[3] S.F. Altschul, T.L. Madden, A.A. Schaer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipma. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, pages 3389–3402, 1997.

[4] A. Bairoch. The enzyme database in 2000. *Nucleic Acids Research*, 28:304–305, 2000.

[5] G.H. Bakir, T. Hofmann, B. Scholkopf, A.J. Smola, B. Taskar, and S.V.N. Vishwanathan, editors. *Predicting Structured Data.* Advances in neural information processing systems. MIT Press, Cambridge, MA, USA, 2007.

[6] A.L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[7] R.E. Barlow, D.J. Bartholomew, J.M. Bremmer, and H.D. Brunk. *Statistical inference under order restrictions; the theory and application of isotonic regression.* Wiley, New York. ISBN 0-4-71-04970-0, 1972.

[8] Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2008.

[9] C. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[10] H. Boeger, D. Bushnell, R. Davis, J. Griesenbeck, Y. Lorch, and J. Strattan et al. Structural basis of eukaryotic gene transcription. *FEBS Letters*, (899-903), 2005.

[11] M.P Brown, W.N. Grundy, D. Lin, N. Cristianini, C.W. Sugnet, and T.S. Furey et al. Knowledge-based analysis of microarray gene expression data by using Support Vector Machines. *Proceedings of National Academy of Science USA*, 97:262–267, 2000.

[12] O. Burdakov, O. Sysoeve, A. Grimvall, and M. Hussain. *Large-Scale Nonlinear Optimization*, pages 25–33. In Nonconvex Optimization and Its Application. Springer-Verlag, Berlin, 2006.

[13] N.A. Christakis and J.H. Fowler. The spread of obesity in a large social network over 32 years. *New England Journal of Medicine*, 357(4):370–379, 2007.

[14] N.A. Christakis and J.H. Fowler. The collective dynamics of smoking in a large social network. *New England Journal of Medicine*, 358(21):2249–2258, 2008.

[15] H.N. Chua, W.K. Sung, and L. Wong. Exploiting indirect neighbors and topological weight to predict protein function from protein-protein interactions. *Bioinformatics*, 22:1623–1630, 2006.

[16] F.R. Chung. *Spectral Graph Theory.* CBMS Regional Conference Series in Mathematics. American Mathematical Society, 1997.

[17] S.R. Collins, P. Kemmeren, X. Zhao, J.F. Greenblatt, F. Spencer, and F.C. Holstege et al. Toward a comprehensive atlas of the physical interactome of *Saccharomyces cerevisiae. Molecular Cell Proteomics*, 6:439–450, 2007.

[18] Gene Ontology Consortium. Gene Ontology: tool for unification of biology. *Nature Genetics*, 25:25–29, 2000.

[19] M. Costanzo, A. Baryshnikova, J. Bellay, Y. Kim, E.D. Spears, and C.S. Sevier et al. The genetic landscape of a cell. *Science*, 327(5964):425–431, 2010.

[20] K. Crammer and Y. Singer. On the algorithmic implementation of multi-class SVMs. *Journal of Machine Learning Research*, 2001.

[21] N. Cristianini, J. Shawe-Taylor, and J. Kandola. On kernel target alignment. *Advances in Neural Information Processing Systems*, pages 367–373, 2002.

[22] J. Davis and M. Goadrich. The relationship between precision-recall and ROC curves. *International Conference on Machine Learning*, 2006.

[23] M. Deng, T. Chen, and F. Sun. An integrated probabilistic mode for functional prediction of proteins. *Journal of Computational Biology*, 11:463–475, 2004.

[24] R. Edgar, M. Domrachev, and A.E. Lash. Gene expression omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Research*, pages 207–210, 2002.

[25] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least Angle Regression. *Annals of Statistics*, 32(2):407–499, 2004.

[26] R. Eisner, B. Poulin, D. Szafron, P. Lu, and R. Greiner. Improving protein function prediction using the hierarchical structure of the gene ontology. *CIBCB Conference*, pages 354–363, 2005.

[27] O. Emanuelsson, S. Brunak, G. von Heijne, and H. Nielsen. Locating proteins in the cell using TargetP, SignalP, and related tools. *Nature Protocols*, 2:953–971, 2007.

[28] R. Finn, J. Mistry, J. Tate, P. Coggill, A. Heger, and J.E. Pollington et al. The Pfam protein families database. *Nucleic Acids Research*, 38:D211–D222, 2010.

[29] J. Gillis and P. Pavlidis. Multifunctionality drives gene characterization. In Submission, 2010.

[30] K.I. Goh, M.E. Cusick, D. Valle, B. Childs, M. Vidal, and A.L. Barabasi. The human disease network. *Proceedings of National Academy of Science USA*, 104:8685–8690, 2007.

[31] H. Yu H, P. Braun, M.A. Yildirim, I. Lemmons, K. Venkatesan, and J. Sahalie. High quality binary protein interaction map of the yeast interactome network. *Science*, 322:104–110, 2008.

[32] B.H. Hall, A.B. Jaffe, and M. Trojtenberg. The NBER patent citation data file: Lessons, insights and methodological tools. *NBER Working Paper 8489*, 2001.

[33] J. Han, N. Bertin, T. Hao, D.S. Goldberg, G.F. Berriz, and L.V. Zhang et al. Evidence for dynamically organized modularity in the yeast proteinprotein interaction network. *Nature*, 430, 2004.

[34] H. Hanley and B. McNeil. The meaning and use of the area under a receiver operator characteristic (ROC) curve. *Radiology*, 1982.

[35] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[36] H Hegyi and M. Gerstein. The relationship between protein structure and function: a comprehensive survey with application to yeast genome. *Journal of Molecular Biology*, 288:147–164, 1999.

[37] M.A. Hibbs, D.C. Hess, C.L. Myers, C. Huttenhower, and O.G. Troyanskaya. Exploring the functional landscape of gene expression: directed search of large microarray compendia. *Genome Biology*, 23(20):2692–2699, 2007.

[38] H. Hishigaki, K. Nakai, T. Ono, A. Tanigami, and T. Takagi. Assessment of prediction accuracy of protein function from protein-protein interaction data. *Yeast*, 18:523–531, 2001.

[39] P. Hu, S.C. Janga, M. Babu, J.J. Diaz-Mejia, G. Butland, and W. Yang et al. Global functional atlas of *Escherichia coli* encompassing previously uncharacterized proteins. *PLoS Biology*, 7:e96, 2009.

[40] W. Huh, J. Falvo, L. Gerke, A. Carroll, R. Howson, J. Weissman, and E.K. O'Shea. Global analysis of protein localization in budding yeast. *Nature*, 425:686–691, 2003.

[41] C. Huttenhower, E.M Haley, M.A. Hibbs, V. Dumeaux, D.R. Barrett, H.A. Coller, and O.G. Troyanskaya. Exploring the human genome with functional maps. *Genome Research*, 19:1093–1106, 2009.

[42] J. Jin, X. Xie, C. Chen, J. Park, C. Stark, D.A. James, and et al. Eukaryotic protein domains as functional units of cellular evolution. *Science Signaling*, 2, 2009.

[43] K. Kanehisa and S. Goto. KEGG: Kyoto Encyclopedia of Genes and Genome. *Nucleic Acids Research*, pages 27–30, 2000.

[44] U. Karaoz, T.M. Murali, S. Letovsky, Y. Zheng, C. Ding, C.R. Cantor, and S. Kasif. Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of National Academy of Science USA*, 101:2888–2893, 2003.

[45] O.D. King, R.E. Foulger, S.S. Dwight, J.V. White, and F.P. Roth. Predicting gene function from patterns of annotation. *Genome Research*, 13, 2003.

[46] R. Kondor and J. Lafferty. Diffusion kernels on graphs and other discrete input spaces. *International Conference on Machine Learning (ICML)*, 2002.

[47] N.J. Krogan, G. Cagney, H. Yu, G. Zhong, X. Guo, and A. Ignatchenko et al. Global landscape of protein complexes in the yeast *Saccharomyces cerevisiae*. *Nature*, 440:637–643, 2006.

[48] G.R. Lanckriet, T. De Bie, N.Cristianini, M.I. Jordan, and W.S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20:2626–2635, 2004.

[49] C.L. Lawson and R.J. Hanson. *Solving Least Squares Problems*. Prentice-Hall, 1974.

[50] I. Lee and E.M. Marcotte. Integrating functional genomics data. *Methods in Molecular Biology*, 453:267–278, 2008.

[51] T.I. Lee, N.J. Rinaldi, F. Robert, D.T. Odom, Z. Bar-Joseph, and G.K. Gerber et al. Transcriptional regulatory networks in *Saccharomyces cerevisiae*. *Science*, 298:799–804, 2002.

[52] C. Leslie, E. Eskin, and W.S. Noble. The spectrum kernel: a string kernel for svm protein classification. *In proceedings of The Pacific Symposium on Biocomputing*, pages 564–575, 2002.

[53] C. Leslie, E. Eskin, J. Weston, and W.S. Noble. Mismatch string kernels for SVM protein classification. *Advances in Neural Information Processing Systems*, 15:1141–1448, 2002.

[54] D.P. Lewis and W.S. Noble. Support Vector Machine learning from heterogeneous data: an empirical analysis using protein sequence and structure. *Bioinformatics*, 22(22):2753–2760, 2006.

[55] C. Lippert, Z. Ghahramani, and K. Borgwardt. Gene function prediction from synthetic lethality networks via ranking on demand. *Bioinformatics*, 26:912–918, 2010.

[56] E.M. Marcotte, M. Pellegrini, M.J. Thompson, T.O. Yeates, and D. Eisenberg. A combined algorithm for genome-wide prediction of protein function. *Nature*, 42:83–86, 1999.

[57] G. Mishra, M. Suresh, K. Kumaran, N. Kannabiran, S. Suresh, and P. Bala et al. Human protein reference database – 2006 update. *Nucleic Acids Research*, 34(Database Issue):D411–D414, 2006.

[58] S. Mostafavi and Q. Morris. Using the Gene Ontology hierarchy when predicting gene function. *Conference on Uncertainty in Artificial Intelligence*, 2009.

[59] S. Mostafavi and Q. Morris. Fast integration of heterogeneous data sources for predicting gene function with limited annotation. *Bioinformatics*, 26:1759–1765, 2010.

[60] 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.

[61] T.M. Murali, C.J. Wu, and S. Kasif. The art of gene function prediction. *Nature Biotechnology*, 24:1474–1475, 2006.

[62] C.L. Myers, DR. Barrett, M.A. Hibbs, C. Huttenhower, and O.G. Troyanskaya. Finding function: evaluation methods for functional genomic data. *BMC Genomics*, 7:187, 2006.

[63] C.L. Myers, D. Robson, A. Wible, M. Hibbs, C. Chiriac, C.L. Theesfeld, K. Dolinski, and O.G. Troyanskaya. Discovery of biological networks from diverse functional genomic data. *Genome Biology*, 6:R114, 2005.

[64] C.L. Myers and O.G. Troyanskaya. Context-sensitive data integration and prediction of biological networks. *Bioinformatics*, 23:2322–2330, 2007.

[65] E. Nabieva, K. Jim, A. Agarwal, B. Chazelle, and M. Singh. Whole-proteome prediction of protein function via graph-theoretic analysis of interaction maps. *Bioinformatics*, 2(Suppl. 1), 2005.

[66] M.E.J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Science USA*, 103:8577–8582, 2006.

[67] M.E.J. Newman. *Networks: An Introduction.* Oxford University Press, 2010.

[68] J. Nocedal and S.J. Wright. *Numerical Optimization.* Springer, 2006.

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

[70] C.S. Ong and A. Zien. An automated combination of kernels for predicting protein subcellular localization. *Proceedings of the 8th Workshop on Algorithms in Bioinformatics (WABI)*, pages 179–186, 2008.

[71] M. Oti and HG. Brunner. The modular nature of genetic diseases. *Clinical Genetics*, 1:1–11, 2007.

[72] X. Pan, D. Yuan, S. Ooi, X. Wang, S. Sookhai-Mahadeo, P. Meluh, and J.D. Boeke. dSLAM analysis of genome-wide genetic interactions in *Saccharomyces cerevisiae*. *Methods*, 41:206–22, 2007.

[73] J. Park and A.L. Barabasi. Distribution of node characteristics in complex networks. *Proceedings of National Academy of Science USA*, 104(46):17916–17920, 2007.

[74] P. Pavlidis, J. Weston, J. Cai, and W.S. Noble. Learning gene functional classification from multiple data types. *Journal of Computational Biology*, 9:401–411, 2002.

[75] M. Pellegrini, E.M. Marcotte, M.J. Thompson, D. Eisenberg, and T.O. Yeates. Assigning functions by comparative genome analysis: protein phylogenetic profiles. *Proceedings of National Academy of Science USA*, 96:4285–4288, 1999.

[76] L. Pena-Castillo, M. Tasan, C.L. Myers, H. Lee, T. Joshi, and C. Zhang et al. A critical assessment of *Mus musculus* gene function prediction using integrated genomic evidence. *Genome Biology*, 9(Suppl 1):S2, 2008.

[77] K. Petersen and M. Pedersen. *The Matrix Cookbook*. Technical University of Denmark, 2008.

[78] J. Platt. *Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods*. Large Margin Classifiers. MIT Press, 1999.

[79] J.C. Platt. Using analytic QP and sparseness to speed the training of Support Vector Machines. *Neural Information Processing Systems (NIPS)*, 1999.

[80] Y. Qi, Y. Suhail, Y.Y. Lin, J.D. Boeke, and J.S. Bader. Finding friends and enemies in an enemies-only network: A graph diffusion kernel for predicting novel genetic interactions and co-complex membership from yeast genetic interactions. *Genome Research*, 18(1991-2004), 2008.

[81] A. Rakotomamonjy, F. Bach, S. Canu, and Y. Grandvalet. SimpleMKL. *Journal of Machine Learning Research*, 2008.

[82] M. Rogers and A. Ben-Hur. The use of Gene Ontology evidence codes in preventing classifier assessment bias. *Bioinformatics*, 25(9):1173–1177, 2009.

[83] H. Rue and L. Held. *Gaussian Markov Random Fields: Theory and Applications*, volume 104 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, London, 2005.

[84] M. Schena, D. Shalon, R.W. Davis, and P. Brown. Quantitative monitoring of gene expression patterns with a complementary DNA microarray. *Science*, 270:467–470, 1995.

[85] B. Scholkopf and A.J. Smola. *Learning with kernels : Support Vector Machines, regularization, optimization, and beyond*. Adaptive computation and machine learning. MIT Press, 2002.

[86] B. Shahbaba and R. Neal. Gene function prediction with hierarchical models with hierarchy-based prior. *BMC Bioinformatics*, 7:448, 2006.

[87] D. Smedley, S. Haider, B. Ballester, R. Holland, D. London, G. Thorisson, and A. Kasprzyk. BioMart–biological queries made easy. *BMC Genomics*, 14:10–22, 2009.

[88] A. Sokolov and A. Ben-Hur. Hierarchical classification of Gene Ontology terms using the GOstruct method. *Journal of Bioinformatics and Computational Biology*, 2010.

[89] J. Song and M. Singh. How and when should interactome-derived clusters be used to predict functional modules and protein function? *Bioinformatics*, 25(23):3143–3150, 2009.

[90] S. Sonnenberg, G. Ratch, C. Schafer, and B. Scholkpof. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 2006.

[91] C. Stark, BJ. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers. BioGRID: a general repository for interaction datasets. *Nucleic Acids Research*, 1(Database issue):D539–D539, 2006.

[92] B. Taskar, C. Guestrin, and D. Koller. Max-Margin Markov networks. *Neural Information Processing Systems Conference*, 2003.

[93] A.H. Tong, G. Lesage, G.D. Bader, H. Ding, H. Xu, and J. Young et al. Global mapping of the yeast genetic interaction network. *Science*, 303:808–813, 2004.

[94] A.L. Traud, E.D. Kelsic, P.J. Mucha, and M.A. Porter. Community structure in online collegiate social networks. *arXiv*, 0809.0690, 2008.

[95] O.G. Troyanskaya, K. Dolinski, A.B. Owen, R.B. Altman, and D. Botstein. A bayesian framework for combining heterogeneous data sources for gene function prediction. *Proceedings of National Academy of Science, USA*, 100:8348–8353, 2003.

[96] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector learning for interdependent and structured output spaces. *International Conference on Machine Learning*, 2004.

[97] K. Tsuda, H.J. Shin, and B. Scholkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21(Suppl 2):ii59–ii65, 2005.

[98] P. Uetz, L. Giot, G. Cagney, T.A. Mansfield, R.S. Judson RS, and J.R. Knight et al. A comprehensive analysis of protein-protein interactions in *Saccharomyces cerevisiae*. *Nature*, 403:623–627, 2000.

[99] A. Vazquez, A. Flammini, A. Martian, and A. Vespignani. Global protein function prediction from protein-protein interaction networks. *Nature Biotechnology*, 21:697–700, 2003.

[100] V. Velculescu, L. Zhang, B. Vogelstein, K., and Kinzler. Serial analysis of gene expression. *Science*, 270:484–487, 1995.

[101] C. von Mering, M. Huynen, D. Jaeggi, S. Schmidt, P. Bork, and B. Snel. STRING: a database of predicted functional associations between proteins. *Nucleic Acids Research*, 31(1):258–261, 2003.

[102] C. von Mering, R. Krause, B. Snel, M. Cornell, S.G. Oliver, S. Fields, and P. Bork. Comparative assessment of large-scale data sets of protein-protein interactions. *Nature*, 417:399–403, 2002.

[103] D. Warde-Farley, SL. Donaldson, O. Comes, K. Zuberi, R. Badrawi, and P. Chao et al. The GeneMANIA prediction server: biological network integration for gene prioritization and predicting gene funciton. *Nucleic Acids Research*, 38(W214-W220), 2010.

[104] J. Weston, A. Elisseeff, D. Zhou, C. Leslie, and W.S. Noble. Protein ranking: From local to global structure in the protein similarity network. *Proceedings of National Academy of Science USA*, 101:6559–6563, 2004.

[105] B. Zhang and S. Horvath. A general framework for weighted gene co-expression network analysis. *Statistical Applications in Genetics and Molecular Biology*, 4, 2005.

[106] L.V. Zhang, O.D. King, S.L Wong, D.S. Goldberg, H.A. Tong, and G. Lesage et al. Motifs, themes and thematic maps of an integrated *Saccharomyces cerevisiae* interaction network. *Journal of Biology*, 4:6, 2005.

[107] D. Zhou, O. Bousquet, J. Weston, and B. Scholkopf. Learning with local and global consistency. *Advances in Neural Information Processing Systems*, 16:321–328, 2004.

[108] X. Zhou, M.C. Kao, and W.H. Wong. Transitive functional annotation by shortest-path analysis of gene expression data. *Proceedings of National Academy of Science USA*, 99(20):12783–12788.

[109] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian Fields and harmonic functions. *The Twentieth International Conference on Machine Learning*, pages 912–919, 2003.

[110] H. Zou and T. Hastie. Regularization and variable selection via the Elastic Net. *Journal of Royal Statistics Society*, 67(2):301–320, 2005.