Positive-Unlabeled Learning in the Context of Protein Function Prediction

By

Noah Youngs

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

September 2014

Dennis E. Shasha

Richard A. Bonneau

Dedication

For Susan Cohen and Robert Youngs, whose nurturing support got me where I am today, gave me a sound platform to stand on in life, and who always believed in me more than I did. And for all the people I have loved and who have loved me along the journey so far. You know who you are, and you make it all worthwhile.

Acknowledgement

There are more people here to thank than I'm sure I will remember, and those that I do deserve more recognition than a handful of words can express, but…

Thanks to Dennis Shasha for his endless insights, gentle encouragement, and the occasional swift kick to the seat of the pants. Thanks to Rich Bonneau for his passion, enthusiasm, and guidance.

Thanks to my parents, Robert Youngs and Susan Cohen, for unwavering support. Thanks to my sister, Sarah Youngs, for always being a stable sounding board.

Thanks to Duncan Penfold-Brown, for always putting in the extra effort, even on projects that weren't technically your job. A lot of this work would not have happened without your help. Thanks to the many mentors, colleagues, and coworkers who have helped foster ideas: Kevin Drew, Glenn Butterfoss, Tim Craven, Doug Renfrew, Emily Koo, and many others. Thanks to my thesis readers, Kris Gunsalus, Zvi Kedem, and Davi Geiger, for making the culmination of this journey possible.

Table of Contents

List of Figures

List of Tables

1. Abstract

With the recent proliferation of large, unlabeled data sets, a particular subclass of semisupervised learning problems has become more prevalent. Known as positive-unlabeled learning (PU learning), this scenario provides only positive labeled examples, usually just a small fraction of the entire dataset, with the remaining examples unknown and thus potentially belonging to either the positive or negative class. Since the vast majority of traditional machine learning classifiers require both positive and negative examples in the training set, a new class of algorithms has been developed to deal with PU learning problems.

A canonical example of this scenario is topic labeling of a large corpus of documents. Once the size of a corpus reaches into the thousands, it becomes largely infeasible to have a curator read even a sizable fraction of the documents, and annotate them with topics. In addition, the entire set of topics may not be known, or may change over time, making it impossible for a curator to annotate which documents are NOT about certain topics. Thus a machine learning algorithm needs to be able to learn from a small set of positive examples, without knowledge of the negative class, and knowing that the unlabeled training examples may contain an arbitrary number of additional but as yet unknown positive examples.

Another example of a PU learning scenario recently garnering attention is the protein function prediction problem (PFP problem). While the number of organisms with fully sequenced genomes continues to grow, the progress of annotating those sequences with the biological functions that they perform lags far behind. Machine learning methods have already

been successfully applied to this problem, but with many organisms having a small number of positive annotated training examples, and the lack of availability of almost any labeled negative examples, PU learning algorithms can make large gains in predictive performance.

The first part of this dissertation motivates the protein function prediction problem, explores previous work, and introduces novel methods that improve upon previously reported benchmarks for a particular type of learning algorithm, known as Gaussian Random Field Label Propagation (GRFLP). In addition, we present improvements to the computational efficiency of the GRFLP algorithm, and a modification to the traditional structure of the PFP learning problem that allows for simultaneous prediction across multiple species.

The second part of the dissertation focuses specifically on the positive-unlabeled aspects of the PFP problem. Two novel algorithms are presented, and rigorously compared to existing PU learning techniques in the context of protein function prediction. Additionally, we take a step back and examine some of the theoretical considerations of the PU scenario in general, and provide an additional novel algorithm applicable in any PU context. This algorithm is tailored for situations in which the labeled positive examples are a small fraction of the set of true positive examples, and where the labeling process may be subject to some type of bias rather than being a random selection of true positives (arguably some of the most difficult PU learning scenarios).

The third and fourth sections return to the PFP problem, examining the power of tertiary structure as a predictor of protein function, as well as presenting two case studies of function prediction performance on novel benchmarks. Lastly, we conclude with several promising avenues of future research into both PU learning in general, and the protein function prediction problem specifically.

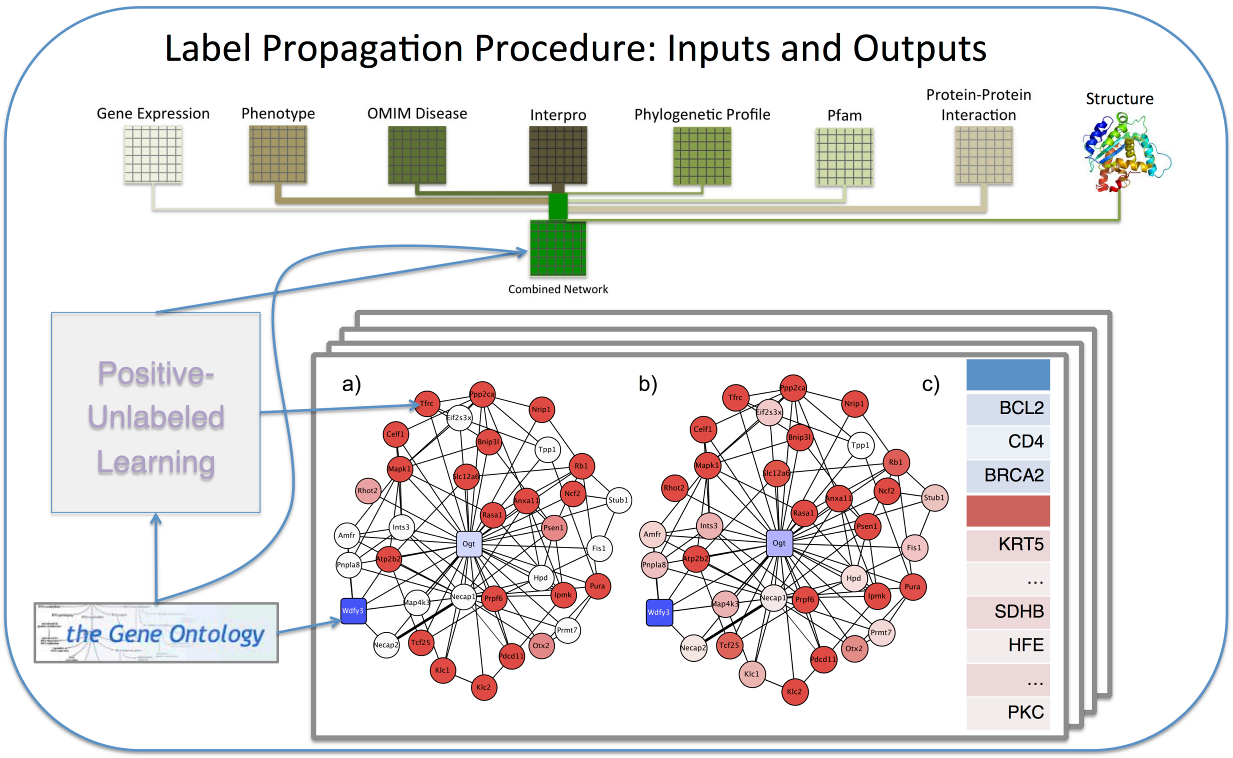


Figure 1.1

A graphical depiction of the label propagation function prediction process. Different datatypes are processed as similarity matrices, and combined utilizing existing Gene Ontology (GO) annotations, as well as negative examples obtained via Positive-Unlabeled (PU) learning. This combined affinity matrix represents a network, different for each function being predicted, where positive and negative labels, as well as prior biases for unlabeled nodes, are again derived from the GO database and PU-learning. Label information is then propagated throughout the network, and predictions are made based on the resulting positivity and negativity of each node.

2. Protein Function Prediction

2.1 Motivation

The rate of new protein discovery has, in recent years, outpaced our ability to annotate and characterize new proteins and proteomes. In order to combat this functional annotation deficit, many groups have successfully turned to computational techniques, attempting to predict the function of proteins to guide experimental verification. Specifically, there has been a surge of interest in applying machine learning methods to the problem of protein function prediction (FP), in order to take advantage of the wealth of biological data available for each protein beyond its sequence, such as computationally-predicted tertiary structure which has already been shown to aid FP (Drew *et al.,* 2011). While traditional approaches to FP mainly involved either homology (with limitations of accuracy) or manual curation (dependent on rare expertise), these new methods present new evaluation and comparative challenges. The MouseFunc competition (Pena-Castillo *et al.,* 2008) was organized to test the ability of machine learning methods to take advantage of large integrated data-sets and provide useful predictions of gene function.

The validity of integrative approaches to function prediction was first demonstrated by the works of Marcotte *et al.* (1999) and Troyanskaya *et al.* (2003), which respectively used linkage graphs and a Bayesian network to predict function. By the time of the MouseFunc competition, FP methods had become quite diverse, including: Support Vector Machines, Random Forests, Decision Trees, and several composite methods (Guan *et al.*, 2008; Lee *et al.,* 2006; Obozinski *et al*., 2008; Tasan *et al.,* 2008), but a recurring theme was to use protein-protein networks of various types to determine function based on guilt-by-association (Kim *et al.,* 2008; Leone *et al.,* 2005; Qi *et al.,* 2008; Zhang *et al.,* 2008). In such a method, genes are represented by nodes in a network, with weighted edges defined by a similarity metric obtained from raw data (often the Pearson Correlation Coefficient of feature vectors). Predictions are then formed by propagating information from genes known to have a function, through the network to unlabeled genes.

While providing unprecedented accuracy, the methods of the MouseFunc competition exposed several general challenges still remaining for the FP problem: 1) choosing a set of high-confidence negative examples, 2) utilizing available data to form prior beliefs about the biological functions of a gene, and 3) effectively combining disparate data sources. As no comprehensive database of functional negative examples currently exists, and nearly all major machine learning methods require a negative class for the training of a classifier, the selection of high-confidence negative examples is especially important for the FP problem.

In this work, we begin to address these challenges by presenting a parameterizable Bayesian technique for computing prior functional biases for each gene, and a novel method for selecting negative examples using these biases. To apply our method, we utilize the framework of the GeneMANIA algorithm (Mostafavi *et al.,* 2008), one of the highest-performing competitors in MouseFunc. In addition to our new priors and negative examples, we present a framework for tuning our Bayesian parameters and other parameters in the original GeneMANIA algorithm. To facilitate this parameter tuning, we incorporate new optimization techniques that take advantage of the structure of the optimization problem. We also integrate our novel negative examples into the GeneMANIA network combination algorithm that synthesizes heterogeneous data into one affinity network.

While well-established procedures exist for the comparison of machine learning methods, recent work (Greene and Troyanskaya, 2012; Pavlidis and Gillis, 2012) has exposed and discussed problems that can be introduced into these comparisons by the nature of biological data. In order to mitigate these biases, we heed the suggestions of Greene and Troyanskaya (2012), and focus on evaluation with a temporal hold out (an evaluation set of annotations obtained at a later point in time than the training data, referred to in this paper and in the MouseFunc competition as the "novel evaluation setting"). We also include one of the few available gold standard evaluation sets (an exhaustive experimental evaluation of the presence of a particular protein function across an entire genome), obtained from Huttenhower et al. (2009). Our goal is to demonstrate the performance improvements of our new algorithm over the existing state-of-the-art in a fair (apples-to-apples) comparison across several datasets. We expect that these comparative results will generalize to other datasets as they become available.

2.2 Previous Work

We present our novel methods using the framework of the GeneMANIA function prediction algorithm of Mostafavi *et al.* (2008), which incorporates prior beliefs and an intelligent network combination algorithm into its guilt-by-association framework. GeneMANIA is a form of Gaussian Random Field (GRF) label propagation, a semi-supervised technique pioneered by Zhou *et al*. (2004); Zhu *et al*. (2003), and provides predictions for genes one function at a time. Given a set of nodes (genes) in a network whose edges define pairwise similarity, and a vector $\vec{y}$ of prior label biases for the nodes given the current function being examined, the GRF algorithm assigns a discriminant value f_i to each node, which can be ranked to produce predictions. The label biases y_i take values in [-1,1], with -1 representing known negative labels, 1 representing known positive labels, and values in between reflecting prior belief about the likelihood of a gene having the function in question. The final discriminant vector $\vec{f}$ is obtained by solving the optimization problem:

(Equation 2.1)

$$\min_f \sum (f_i - y_i)^2 + \sum\sum \mathbf{W}_{\boldsymbol{ij}}\left(f_i - f_j\right)^2$$

This equation has an analytical solution in the form of a linear system: $\mathbf{A}x = b$, and also guarantees that the discriminant values f_i will lie in the range [-1,1], with larger values indicating greater likelihood of an unlabeled node being a positive example of the function in question.

The analytical solution for the discriminant vector $\vec{f}$ mentioned above takes the following form:

$$(\mathbf{I} + \mathbf{L})\vec{f} = \vec{y}$$

With $\mathbf{L} = \mathbf{D} - \mathbf{W}$**,** where **I** is the identity matrix, $\vec{y}$ is the vector of prior beliefs, W the pairwise similarity matrix obtained by integrating multiple data types, and **D** is a diagonal matrix with

$\mathbf{D}_{ii} = \sum_j \mathbf{W}_{ij}$ This solution clearly only requires solving a linear system of the form $\mathbf{A}x = b$, and with proper normalization of **W**, A, is guaranteed to be symmetric positive-definite. Thus the conjugate gradient algorithm can speedily and reliably solve for the discriminant vector $\vec{f}$.

Intuitively, this algorithm allows prior information to flow through the network until equilibrium is reached. The objective function propagates known labels through the similarity network via the second ``smoothness" term in Equation 2.1, weighted by the strength of similarity between nodes as specified by the network, and also enforces adherence to the prior bias through the first ``consistency" term in Equation 2.1. Thus the label biases, both the positive and negative examples as well as biases used for unlabeled nodes, play a very important role in the algorithm. Mostafavi and Morris (2009) explore variations on techniques to choose the label bias vector, but we expand upon this work to improve accuracy in our algorithm by utilizing more of the information contained in current functional annotations to determine functional biases and negative examples (see section 2.3.1).

The other key component of the GRF algorithm is the composite network defining similarity between all pairs of genes. Mostafavi *et al.* (2008) proposed a method to combine disparate data sources, each represented as an affinity matrix, into one composite matrix, based on the work of Tsuda *et al.* (2005). This algorithm, for each Gene Ontology (GO) functional terms of interest, maximizes the similarity between pairs of positively labeled genes and minimizes the similarity between genes of opposite labels. This is accomplished by calculating the final network $\mathbf{W}^{\bullet}$ as the weighted sum of each individual network $\mathbf{W}_i$, with the vector of weights, $\hat{\alpha}^{\bullet}$ chosen to solve:

(Equation 2.2)

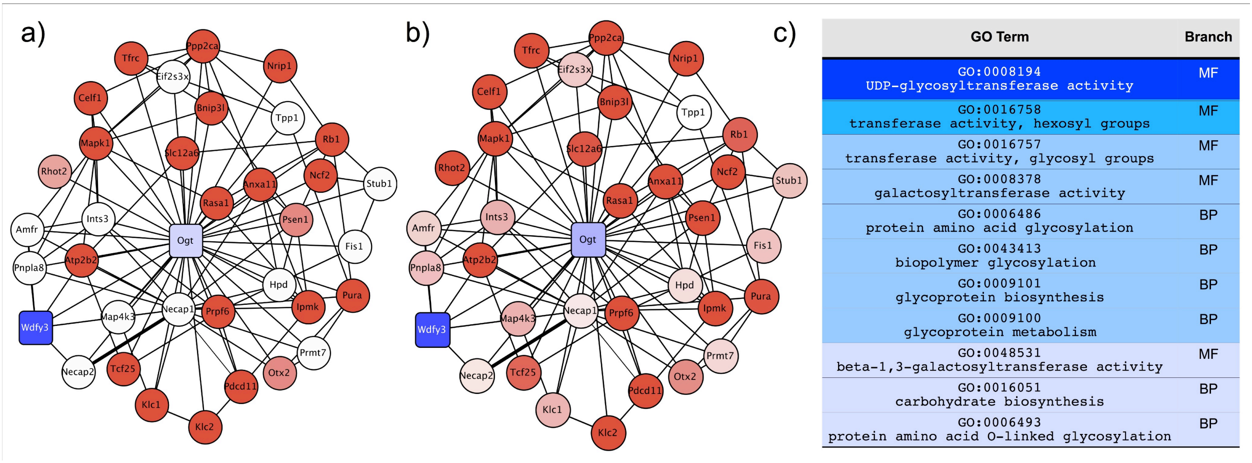
$$\hat{\alpha}^{*} = \min_{\alpha}\left((\boldsymbol{\Omega}\hat{\alpha} - \hat{t})^{\intercal}(\boldsymbol{\Omega}\hat{\alpha} - \hat{t})\right)$$

If there are *n* nodes, *p_l* positively labeled nodes, *n_l* negatively labeled nodes, and *h* different data types, then $\boldsymbol{\Omega}$ is a $(p_l^2 + p_l n_l) \times h$ matrix, where each column contains all of the entries in $\mathbf{W}_i$ corresponding to the positive-positive and positive-negative label pairs. The target vector $\hat{t}$ contains the values: ${n_l^2}/{n^2}$ for positive-positive pairs and ${-n_l p_l}/{n^2}$ for positive-negative pairs, in order to deal with class imbalance in the labeled data. The resulting vector $\hat{\alpha}^{*}$ will have length equal to the number of different data-types (possibly with some zero entries to signify data that was not discriminative) and contain the relative importance of each data-type determined by the algorithm.

This network combination algorithm is prone to over-fitting in cases with few positive examples. The original GeneMANIA algorithm addressed this problem by introducing a regularization term, but later work (Mostafavi and Morris, 2010) instead attempts to fit the composite data network for multiple Gene Ontology (GO) terms simultaneously. Our algorithm expands upon this second approach by directly incorporating our negative examples (see section 2.3.3).

2.3 Novel Methods

We propose novel techniques focusing on several key aspects of protein function prediction: choosing negative examples, forming label biases for unlabeled genes with some known annotations, and an issue specific to GRF-based methods, namely combining heterogeneous data-types into one affinity network. In addition we suggest a new optimization algorithm tailored to our techniques, and provide a framework for tuning parameters using the training data.



GO Term	Branch
GO:0008194 UDP-glycosyltransferase activity	MF
GO:0016758 transferase activity, hexosyl groups	MF
GO:0016757 transferase activity, glycosyl groups	MF
GO:0008378 galactosyltransferase activity	MF
GO:0006486 protein amino acid glycosylation	BP
GO:0043413 biopolymer glycosylation	BP
GO:0009101 glycoprotein biosynthesis	BP
GO:0009100 glycoprotein metabolism	BP
GO:0048531 beta-1,3-galactosyltransferase activity	MF
GO:0016051 carbohydrate biosynthesis	BP
GO:0006493 protein amino acid O-linked glycosylation	BP

Figure 2.1 a) A subsection of the association network before the algorithm is run, showing prior beliefs for genes for the function: GO:0008194, UDP-glycotransferase activity, focusing on gene "Ogt". The shading of the nodes represents the degree of positivity compared to the mean of all prior biases, with blue indicating greater likelihood of possessing the function in question, red lesser likelihood, and white representing genes that had no GO term annotations to use for a prior. Square nodes represent validated true positives (including the training positive example "Wdfy3"). (b) The same subsection of the association network as (a), but after label propagation,

showing the final discriminant values of the genes. (c) The GO terms that are most predictive of the functional term GO:0008194, with darker shades of blue representing stronger predictors.

2.3.1 Label Biases

Mostafavi and Morris (2009) showed that significant performance gain could be achieved by allowing existing GO annotations to inform the priors applied to genes in GRF function prediction, using a technique called Hierarchical Label Bias (HLBias). This idea is supported by the work of King et al. (2003), which showed that patterns of GO annotations alone provided enough signal to predict future annotations. HLBias specified that genes which possessed annotations for functions ancestral to the function of interest received a prior bias equal to the proportion of genes with the ancestral function that also are known to have the function in question.

However, due to the difficulty of defining a functional hierarchy, the structure of the GO tree is often altered by its curators, with terms being moved to different parents, virtually guaranteeing that there exist functional relationships that are non-ancestral. When considering the complexity of functional interactions, it would seem likely that the presence of some functions might influence the likelihood of a gene possessing other functions, regardless of whether or not the relationship between the two is ancestral. This is especially true when considering annotations in all three branches of the GO hierarchy simultaneously. Accordingly, we extend HLBias to include the likelihood of a given function co-occurring with all other existing annotations (across all three branches of the GO Tree: Biological Process, Molecular Function, and Cellular Component), in the following manner:

Let $\hat{p}(c|m)$ denote the empirical conditional probability of seeing annotation *c* given the presence of annotation *m:*

(Equation 2.3)

$$\hat{p}(c|m) = {n_{mc}^{+}}/{n_{m}^{+}}$$

Where n_{mc}^{+} is the number of gene products where both *m* and *c* appear and n_{m}^{+} is the number of gene products that have annotation *m* For a protein *i,* let D_i be the set of all GO terms annotated to *i*. For a given functional term *c*, we approximate the conditional prior probability of gene *i* having function *c* by the score:

(Equation 2.4)

$$prior_i = \frac{1}{|D_i|} \sum_{m \in D_i} \hat{p}(c|m)$$

The label biases are then scaled to the range [-1,1] : $y_i = 2prior_i - 1$

Due to the hierarchical nature of GO terms, some of the conditional probabilities in this calculation will contain redundant information, and so when considering a protein with annotations D_i, we remove from D_i all GO terms which have a child in D_i leaving a set of only the most specific annotations of protein *i* to use in calculating the bias.

Figure 2.1.c provides an example of the most predictive GO terms for the GO functional term UDP-glycotransferase activity (UDPGA), which include many terms which have no ancestral relationship to UDPGA. Examining a specific prediction example, we find that the annotations informing the prior bias for gene Ogt (pictured in Figure 2.1.a), are all non-ancestral terms, and contribute to the algorithm making a correct positive prediction (Figure 2.1.b).

Lastly, we observe a large bias introduced by terms with small sample size, where one term appears to be a perfect predictor of another. In order to reduce the potential for overfitting stemming from this phenomenon, we introduce a weighted pseudocount into the calculation of the empirical conditional probability, whereby equation 2.3 is replaced by:

(Equation 2.5)

$$\hat{p}(c|m) = \frac{n_{mc}^{+}}{n_m^{+} + \gamma e^{\lambda n_m^{+}}}$$

This idea is motivated by the hypotheses that no two GO terms 'c1' and 'c2' should both appear in every protein where one appears, unless 'c1' and 'c2' have an ancestral relationship, and also that the number of undiscovered occurrences of a function is related to the number of currently known occurrences. This equation (via the two parameters γ and λ) allows us to smoothly transition between two extreme assumptions about how missing and currently known annotations are distributed: 1) the number of observations in the data is a proxy for how well a function has been studied, and so the number of missing counts in the data should be inversely proportional to the number already seen, and 2) the number of currently known occurrences is in fact a better representation of the specificity of a function, and so the undiscovered occurrences should be directly proportional to the number already seen.

In order to allow the data itself to choose one of these hypotheses, we sample from a range of combinations of parameters, including the potential for no pseudocounting:

$\gamma \in$ [0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

$\lambda \in$ [-0.05, -0.025, -0.0125, -0.00625, -0.003125, 0, 0.003125, 0.00625, 0.0125, 0.025, 0.05]

Where γ and λ define our label bias pseudocount in equation 2.5. Including $\lambda = 0$ in the range allows for the potential for a constant pseudocount (or no pseudocount at all if $\gamma = 0$).

The final value of the parameters is chosen by tuning with cross-validation over the training set, as described in section 2.3.5.

For genes with no previous annotations in GO, we follow Mostafavi et al. (2008) and set the label bias to the mean of all the label biases calculated for genes with GO annotations, including the positive and negative example genes with values of {1,-1} respectively. We refer to our label bias algorithm hereafter as ALBias.

2.3.2 Negative Examples

The choice of negative training examples for use in supervised machine learning algorithms is a recurring problem for FP methods. While the GO database does include negative annotations, the number of such annotations is currently small. Thus it is necessary to infer negative examples for each function (typically using a heuristic). Past heuristics include (i) designating all genes that don't have a particular label as being negative for that label (Guan *et al.*, 2008), (ii) randomly sampling genes and assuming the probability of getting a false negative is low (often done when predicting protein-protein interactions, as in Gomez et al., 2003), and (iii) using genes with annotations in sibling terms of the term of interest as negative examples

(Mostafavi and Morris, 2009). Mostafavi and Morris (2009) note in discussion that this last technique may often break down as some genes are annotated to more than one sibling term, and many genes have few siblings to use.

We present a new technique for choosing negative examples based upon the label biases calculated for each function. Namely, all genes with an annotation in the same branch of GO as the term being predicted, and which have a $prior_i$ score of 0 for the function in question (with the prior score computed across all three branches of GO), are treated as negative examples for that function. Intuitively, this amounts to treating a gene 'g' as a negative for annotation 'c' if no annotation 's' among the most specific annotations of 'g', ever appeared alongside annotation 'c' in any other gene. (Note that the choice of pseudocounting parameters does not impact the negative examples, as only the magnitude of the label bias will be affected and not whether a bias is non-zero.)

Restricting the negative examples to having an annotation in the same branch as the GO term being predicted, rather than simply having an annotation in any branch, does decrease the number of negative examples, but also more significantly decreases the number of validated true positives that were misclassified as negatives. The number of negatives in mouse decreased by 14.9% due to this restriction, while the number of verifiable misclassified negatives dropped by 22.5%, and in yeast the number of negative examples decreased by 23.2% while the verifiable misclassified negatives dropped by 91.5%.

2.3.3 Network Weighting

As mentioned in our description of previous work, one essential component of the GRF algorithm is synthesizing heterogeneous data sources into one pairwise affinity matrix. Mostafavi and Morris (2010) found that fitting this matrix for multiple GO terms simultaneously significantly decreased overfitting, especially in low-annotation terms. The authors simplified the calculation of this simultaneous fit by considering negative-negative pairs of labels as well as the positive-positive and positive-negative pairs utilized by the original network-weighting algorithm of Mostafavi et al. (2008). This simplification also requires the treatment of all non-positive genes as negative genes for each GO.

As described in Mostafavi and Morris (2010), the Simultaneous Weights (SW) algorithm, which fits network weights to multiple GO terms at the same time, operates by solving the equation:

(Equation 2.6)

$$\hat{\alpha}^{*} = \min_{\alpha}(-2\hat{\alpha}^{\intercal}\boldsymbol{\Omega}\hat{t}^{\intercal} + -2\hat{\alpha}^{\intercal}\boldsymbol{\Omega}^{\intercal}\boldsymbol{\Omega}\hat{\alpha})$$

with $\alpha_i \geq 0$, and $\hat{t} = \sum_{c=1}^{h} \hat{t}_c$ where each c is a different GO term in the same branch of the hierarchy. This equation is a simplification of the formulation:

(Equation 2.7)

$$\hat{\alpha}^{*} = \min_{\hat{\alpha}}(\sum_{c=1}^{h}(\boldsymbol{\Omega_c}\hat{\alpha} - \widehat{t_c})^{\intercal}(\boldsymbol{\Omega_c}\hat{\alpha} - \hat{t}_c)))$$

This simplification is made possible by considering negative-negative pairs of labels as well as the positive-positive and positive-negative pairs, and by treating all non-positive nodes as negative nodes, which causes all Ω_c to be identical, and all $\hat{t}_c$ vectors to be the same length.

Mostafavi and Morris (2010) showed that these simplifications do not hamper performance, and also found that fitting the combined network to all GO terms in a particular branch (GO-BP, GO-CC, or GO-MF) worked better than any other subset or grouping of functions. We concur that fitting to all terms performs better than any of the subsets we attempted, but propose that the apparent indifference of this algorithm to the assumption that all non-positive nodes are negative was most likely due to a lack of any satisfactory alternative for choosing negative examples.

We return to the unsimplified version of the simultaneous fit proposed by Mostafavi and Morris (2010), and utilize our more-specific negative examples that are unique to each GO term. Our modification to the network combination algorithm relies on returning to the formulation of 2.7 but also maintains the unique Ω_c matrices dependent upon both the positive labels and the specific negative examples chosen for term c. Such a formulation still has an analytical solution:

(Equation 2.8)

$$\sum_c \Omega_c^{\mathsf{T}} \hat{t}_c = \sum_c (\Omega_c^{\mathsf{T}} \Omega_c)\alpha^*$$

which can be efficiently solved with a Cholesky decomposition, as $\sum_c(\Omega_c^{\mathsf{T}} \Omega_c)$ is positive definite and is only $h \times h$ in dimension, with h being the number of data sets to be combined. We refer to our network algorithm as Simultaneous Weights with Specific Negatives (SWSN), and note that while the SW algorithm is run on all terms with less than 300 annotations, we fit our SWSN algorithm only to the set of terms where function prediction is to be performed (between 400-500 terms for each benchmark). Our experiments indicate that this reduction in the number of terms fit has a negligible impact on any of the performance metrics.

2.3.4 Successive Block Conjugate Gradient Optimization

The network-weighting scheme defined above creates a single combined matrix W for all functional terms within the same GO branch. Thus the coefficient matrix is identical for the optimization problem that is solved for each function, and so we are faced only with the issue of a different right-hand-side (RHS) per function. In such cases, computational costs can be decreased by methods that solve all of the problems simultaneously, rather than iteratively solving each problem without using any of the information obtained by other solutions. We propose a modified version of the Successive Block Conjugate Gradient algorithm (SBCG) of Suarjana and Law (1994).

In this algorithm, the search direction is obtained simultaneously for all of the distinct RHS vectors in the problem. If at any point the search direction matrix becomes rank-deficient, dependent RHS vectors are moved to a secondary system, but are still updated with steps obtained from the search direction in the primary system, and so still proceed towards convergence. The speed of this secondary convergence is dependent on the angle between the vectors in the primary system and secondary system.

Our algorithm differs from the original proposed by Suarjana and Law (1994) in several ways. Firstly, not all solutions converge to the desired tolerance in the same number of iterations, and so we save computation by removing already-converged RHS vectors from the block calculation rather than updating the entire system until all RHS vectors converge. Secondly, when the RHS vectors in the secondary system are nearly orthogonal to those in the primary system, waiting for secondary convergence can require a large number of iterations. Instead, once all primary system RHS vectors are converged, we restart the algorithm in a second phase with the secondary system as the primary system, but using the latest residuals as our starting point. Lastly, empirical observation has shown some low condition numbers can occur in the secondary phase when the number of dependent RHS vectors is large. We find that splitting up the total number of RHS vectors into a few smaller blocks alleviates this problem without significantly increasing computational cost. For the problem at hand, we chose to divide the function prediction problems into subproblems with a maximum of 500 RHS vectors.

Pseudocode for our adaptation of SBCG is found in algorithm SA1.

Algorithm SA1 SBCG Algorithm, solving $\mathbf{AX} = \mathbf{B}$
$\mathbf{R}$ represents the matrix of residuals, while the m, s and c superscripts denote the primary, secondary, and converged set of residual vectors

```
Initialize: $k = 0$; $\mathbf{R_0} = \mathbf{B} - \mathbf{AX_0}$
Let $\mathbf{R^m} = \mathbf{R_0}$, $\mathbf{R^s} = \{\}$, $\mathbf{R^c} = \{\}$
Let $\mathbf{X^m} = \mathbf{X_0}$, $\mathbf{X^s} = \{\}$, $\mathbf{X^c} = \{\}$
Let $col\{\mathbf{X}\}$ = the number of columns in X
while $col\{\mathbf{R^c}\} < col\{\mathbf{B}\}$ do
    while $col\{\mathbf{R^m}\} > 0$ do
        $k = k + 1$
          % Update search direction
        if $k = 1$ then
            $\mathbf{P_1} = \mathbf{R_0}$
        else
            Solve: $(\mathbf{R^m_{k-2}})^\mathsf{T}\mathbf{R^m_{k-2}}\beta = (\mathbf{R^m_{k-1}})^\mathsf{T}\mathbf{R^m_{k-1}}$
            $\mathbf{P_k} = \mathbf{R_{k-1}} + \mathbf{P_{k-1}}\beta$
        end if
        Orthonormalize $\mathbf{P_k}$, identify dependent indices d
          % Move dependent vectors to secondary system
        $\mathbf{R^m} = \mathbf{R^{m\setminus d}}$,   $\mathbf{R^s} = \mathbf{R^{s\cup d}}$
        $\mathbf{X^m} = \mathbf{X^{m\setminus d}}$,   $\mathbf{X^s} = \mathbf{X^{s\cup d}}$
          % Solve for search direction and steplength
        $\mathbf{U_k} = \mathbf{AP_k}$
        Solve: $\mathbf{P_k^\mathsf{T}U_k}[\alpha^\mathbf{m}_\mathbf{k}, \alpha^\mathbf{s}_\mathbf{k}] = [(\mathbf{R^m_{k-1}})^\mathsf{T}\mathbf{R^m_{k-1}}, (\mathbf{R^s_{k-1}})^\mathsf{T}\mathbf{R^m_{k-}}$
          % Update iterates
        $[\mathbf{X^m_k}, \mathbf{X^s_k}] = [\mathbf{X^m_{k-1}}, \mathbf{X^s_{k-1}}] + \mathbf{P_k}[\alpha^\mathbf{m}_\mathbf{k}, \alpha^\mathbf{s}_\mathbf{k}]$
        $[\mathbf{R^m_k}, \mathbf{R^s_k}] = [\mathbf{R^m_{k-1}}, \mathbf{R^s_{k-1}}] - \mathbf{U_k}[\alpha^\mathbf{m}_\mathbf{k}, \alpha^\mathbf{s}_\mathbf{k}]$
          % Remove converged columns
        for all $i \in m \cup s$ do
            if $\|(\mathbf{R^m_k})_i\| < \epsilon$ then
                if $i \in m$ then
                    $\mathbf{R^m} = \mathbf{R}^{\mathbf{m}\setminus i}$,   $\mathbf{R^c} = \mathbf{R}^{\mathbf{c}\cup i}$
                    $\mathbf{X^m} = \mathbf{X}^{\mathbf{m}\setminus i}$,   $\mathbf{X^c} = \mathbf{X}^{\mathbf{c}\cup i}$
                else
                    $\mathbf{R^s} = \mathbf{R}^{\mathbf{s}\setminus i}$,   $\mathbf{R^c} = \mathbf{R}^{\mathbf{c}\cup i}$
                    $\mathbf{X^s} = \mathbf{X}^{\mathbf{s}\setminus i}$,   $\mathbf{X^c} = \mathbf{X}^{\mathbf{c}\cup i}$
                end if
            end if
        end for
    end while
      % If we have unconverged secondary columns, restart
    $\mathbf{R_0} = \mathbf{R^s}$, $\mathbf{X_0} = \mathbf{X^s}$, $k = 0$
    $\mathbf{R^m} = \mathbf{R_0}$, $\mathbf{R^s} = \{\}$
    $\mathbf{X^m} = \mathbf{X_0}$, $\mathbf{X^s} = \{\}$
end while
```

2.3.5 Parameter Tuning

The multiple RHS framework described in 2.3.4 lends itself well to parameter tuning, as the different combinations of the parameters λ and γ described in 2.3.1 simply yield more RHS label bias vectors to solve for with the same coefficient matrix.

The original formulation of the GRF objective function in Zhou *et al.* (2004) included the parameter μ, which describes the relative weight to be placed on each component of the objective function, which we formulate as:

(Equation 2.9)

$$\min_{f} \sum (f_i - y_i)^2 + (1-\mu) \sum \sum \mathbf{W}_{ij} \left(f_i - f_j\right)^2$$

This parameter was ignored by the GeneMANIA algorithm, but we reintroduce it here, and test its impact on function prediction by adding it to our tuning methodology. We test values of $\mu \in$ [.05, .1, .15, ..., .9, .95].

In order to choose performance-maximizing parameters, we create a synthetic learning problem from the training data, which is characteristically similar to the original learning problem, and choose parameters that yield the best performance on this subproblem.

In order to tune the various parameters for our algorithm, we must create a subset within the training data to measure the performance of different parameter combinations. We begin by subdividing the training data into a tuning subset and a validation subset, with sizes of 3/4 and 1/4 of the training data respectively. We also ensure that the proportion of genes with any GO annotations to those that are completely unannotated are the same in each subset, to preserve the similarity of the training environments. Next, we must adjust the tuning subset to be representative of the original learning scenario.

For the test scenario, such a task is trivial, as all annotations are removed for validation genes that are unlabeled. For the novel scenario, however, the task is more difficult, as the novel scenario in general involves predicting functions for genes which may already have some annotations. We address this issue with the following algorithm:

After splitting out training data into a training and validation subset, we create a novel-like tuning environment by removing a random subset of annotations from a smaller subset of genes in the validation subset, as well as completely removing annotations for some of the validation genes, to simulate a non-systematic addition of partial annotations. The eliminated annotations are then used to evaluate the performance of the algorithm on the training subset of the training data. Any terms where no annotations were removed from the validation subset are deleted from the list of terms to be predicted. The final result is a set of training and validation data derived entirely from the original training data,

which are similar to the final learning problem for the novel mouse and novel yeast evaluation scenarios. Pseudocode for this procedure is presented in Algorithm SA2.

Algorithm SA2 Synthetic Novel Set Generation

```
Separate training data into two sets: $\Upsilon$ and $\Psi$
Let $\Phi$ be a set with the same genes as $\Psi$, but with no annotations
Define $\omega, \nu \in \{0,1\}$
  % First remove all annotations from a subset of genes
for $i = 1 \to \omega * |\Psi|$ do
 Set the annotations for gene $\Phi_i$ equal to all annotations for $\Psi_i$
 Remove all annotations for gene $\Psi_i$
end for
  % Now remove partial annotations from a subset of genes
for all GO terms $g$ present in $\Psi$ do
 Let $n$ = the count of term $g$ in both $\Phi$ and $\Psi$
 Let $l = n*\nu$ - (count of term $g$ in $\Psi$)
   for $i = 1 \to l$ do
      Choose random $j$ s.t. gene $\Psi_j$ has annotation $g$
      Let set c contain $g$ and all children of $g$ annotated to $\Psi_j$
      Add annotations c to $\Phi_j$
      Remove annotations c from $\Psi_j$
   end for
end for
  % Now run predictions on sets $\Upsilon$ and $\Psi$
  % Validate with annotations present in $\Phi$
```

It is often the case that different combinations of parameters will perform better when evaluated by some metrics, and worse when evaluated by others. Combining evaluation metrics into one score proves difficult, as the same magnitudes of difference between the scores of different combinations of parameters does not have the same meaning in different metrics (for example a move in AUC_{ROC} from .97 to .98 is far more significant than a move in AUC_{PR} from .10 to .11, or even a move in *AUC_ROC* from .50 to .51), and a normalization scheme would create dependency on which sets of parameters were selected to evaluate. We choose to define our parameter score as the average of the TopScore metrics (see section 2.4.4), and choose the combination of parameters that maximizes this score. We find that this choice of parameter score improves performance across other metrics as well.

When applying the SBCG algorithm, described in section 2.3.4, to the parameter tuning problem, we find a large amount of rank deficiency among the label biases from the different combinations of parameters in our candidate value sets. Therefore we prefer to apply SBCG longitudinally, solving for all functions at once with a particular set of parameters, rather than solving for all combinations of parameters for a particular function. The greatest performance gain would undoubtedly stem from a framework where all blocks are solved simultaneously in one large system, but we did not explore this option.

2.4 Methods

2.4.1 Evaluation Datasets

We evaluate our algorithm on three datasets: the MouseFunc benchmark, yeast data, and a gold standard data set in yeast. With regard to MouseFunc data, we focus on the Molecular Function branch of the Gene Ontology (GO) hierarchy. For fair comparison to prior work, we use only data available to participants at the time of the MouseFunc exercise: these data include 10 networks (Interpro data, PFAM data, 3 Gene Expression networks, PPI data, Phenotype, 2 Conservation Profile networks, Disease Association data), 1874 molecular function terms, and 21603 mouse genes, with all data gathered in 2006 (see Pena-Castillo *et al*., 2008). Predictions are made, as in MouseFunc, only for functional terms with between 3 and 300 annotations in the genome, but all functional terms are used in the bias calculation and negative example choice described in sections 2.3.1 and 2.3.2.

For our performance evaluation in yeast we focus on the Biological Process branch of the GO tree, using data obtained from Mostafavi and Morris (2010), which includes 44 networks of data obtained from BIOGRID (Stark *et al.,* 2006), covering 3904 genes with 1188 biological process terms (terms with between 3 and 300 annotations). We augment this yeast data with experimentally confirmed gold standard annotations in the BP term of GO:0007005, mitochondrion organization and biogenesis (MOB), obtained from Huttenhower *et al.* (2009) (see section 2.4.3).

2.4.2 Functional Association Data

Association networks were created from feature-based data types using the Pearson Correlation Coefficient, after a frequency transform as described in Mostafavi *et al*. (2008). Only the top 100 interactions are used for each gene in the training set to keep the networks sparse, and a normalization scheme of $W_h' = D_h^{1/2} W_h D_h^{1/2}$ is applied to each network and to the final combined network, where D_h is again the diagonal matrix containing the row sums of W_h.

2.4.3 Evaluation Frameworks

We categorize protein function through Gene Ontology annotations, observing the common convention of excluding ‘Inferred From Electronic Annotation’ (IEA) annotations.

As in the MouseFunc competition, performance is evaluated in two different scenarios: 1) a test set where all GO annotations are removed from a subset of data (1718 genes in mouse) and then predictions are made from the remaining training data, and 2) a novel set where predictions are made for proteins that have received new annotations at a later date in time. The member genes of this second set consist of the intersection of all proteins that have received at least one new annotation in any of the GO terms for which we are attempting predictions (1954 genes in mouse, 362 genes in yeast), and so include many proteins that already had some annotations in the training set, as well as proteins with no annotations in the training set.

We treat the novel scenario as the more important evaluation scheme for this work, because we believe it better reflects the true task facing computational biologists, and is less prone to evaluation biases (Greene and Troyanskaya, 2012). The test set approach suffers from biases stemming from the underlying use of sequence-similarity methods in both input data and GO labeling, which likely explains the better performance of all algorithms in the test scenario vs. the novel scenario.

Pena-Castillo et al. (2008) remarks that there appears to be a qualitative difference between the two types of evaluations used in MouseFunc: the test set (a manually selected leave-out set of genes where all known GO annotations are removed), and the novel set (a set of genes that have acquired new GO annotations at a later time period than the training data). We find that indeed the performance of all algorithms is markedly higher on the test set than on the novel set (see section 5.1 of the main text). This dichotomy in performance is mirrored in later work by

Mostafavi and Morris (2009), which uses the same evaluation setup on more recent GO data. We hypothesize two different factors underlying the relative strength of test performance compared to novel performance:

The first is the fluidity of the Gene Ontology itself. Annotations are not set in stone, and can be added and deleted depending upon further review of the evidence. The structure of the hierarchy is also mutable, with further annotation changes caused by re-structuring as annotations from old ancestors are deleted and new ancestors added to ensure the true-path rule is honored. In summary, GO annotations change significantly over time, causing performance degradation in predictions that span a large temporal gap, such as in the MouseFunc novel evaluation scenario.

The second possible factor lies in the interdependence of both the data and the annotations on sequence-similarity-based methods. Several of the included data-types: Pfam, Interpro, OMIM, etc. use strong sequence similarity to propagate data amongst proteins. The same is true of GO annotations, where computational predictions can be assigned based on homologues after manual review. Thus even if the annotations are entirely removed for the test set, the sequence-similarity links underlying those annotations are still present in the data and thus make the annotations more easily recoverable. This interdependence would be less true in the novel scenario, as one would expect that most of the homologues known at the time that the training data was gathered would have already been annotated in GO as well, and so the majority of the novel annotations likely come from experimental evidence or from newer sequence-similarity searches that are not reflected in the training data.

Despite the biases mentioned above, error results are presented for the test scenario as well, to facilitate comparison with MouseFunc algorithms. For both the novel and test MouseFunc evaluations, predictions are made for the same set of GO Molecular Function terms as the original competition: 488 and 442 terms respectively. For yeast, we show results only in the novel scenario, with data from June 2007, one year after the training data, which includes 511 GO BP terms with at least one new annotation.

Any comparison of computational methods using GO annotations as the ground truth suffers from the lack of delineation between negative and absent annotation. This drawback is discussed at length in Huttenhower *et al*. (2009), and can create significant difficulty in evaluating computational prediction methods, as observed false positive predictions may simply be a function of a lack of study rather than incorrect prediction. It is for this reason that performance evaluation in the novel scenario ignores any false positives for genes outside the novel set, as it likely that these genes were not studied at all in the time interval between the annotation date for training and for testing. In order to further alleviate some of the uncertainty caused by incomplete annotation, we present performance evaluation metrics on a ``gold standard" benchmark of yeast genes experimentally verified by Huttenhower et al. 2009 for GO:0007005 mitochondrial organization and biogenesis (MOB). These gold standard annotations include 148 additional positive annotations that match genes in our yeast gene set, and are also added to the novel set used for the general yeast benchmark. Lastly, when calculating performance statistics on the MOB gold standard, we add an additional 2473 genes to the 342 comprising the yeast novel set. These additional genes are the negative examples from

Huttenhower *et al.* (2009) that are present in our gene set.

2.4.4 Evaluation Metrics: PR vs. ROC, TopScore

The performance of discriminant-based classification algorithms is most often represented by two plots: The Receiver-Operator Characteristic (ROC) curve, and the Precision-Recall (PR) curve, each of which can be summarized by their AUC, the area that the curve encompasses. While both performance measures attempt to describe how well the ordering of discriminant values captures the true positive and negative labels, each has different strengths and weaknesses. Precision tends to be more easily interpretable for an experimentalist, but averaging AUC_{PR} numbers over many classifiers can be misleading due to the nonlinear nature of precision scores (See Figure 2.2).

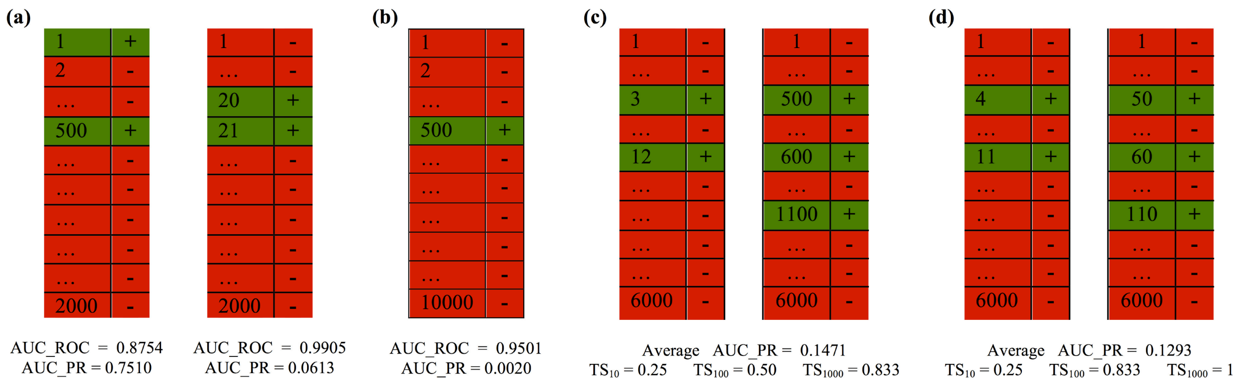


Figure 2.2 (a) AUC_{PR} and AUC_{ROC} scores for an excellent/poor and a mediocre/mediocre ranking of two true positives amidst 1,998 true negatives. (b) High AUC_{ROC} score of a poor ranking of one true positive amidst 9,999 true negatives. (c) Average AUC_{PR} and TopScore

values for an excellent classifier of 2 true positives amongst 5,998 true negatives, and and a poor classifier of 3 true positives amongst 5,997 trues negatives. (d) Average AUC_{PR} and TopScore values for an excellent classifier of 2 true positives amongst 5,998 true negatives, and a mediocre classifier of 3 true positives amongst 5,997 true negatives.

Conversely AUC_{ROC} provides a better global view of the rankings, but lacks a meaningful interpretation for experimentalists and its magnitude depends on the skew of the dataset. See the Supplementary Materials for a more detailed description of the pitfalls of each metric.

When presented with computational predictions, experimentalists must determine the number of predictions to assay, as well as which functions to focus on... a task made more difficult by complicated performance metrics. In order to create a metric more robust to averaging than PR, but which still enjoys easy interpretability for experimentalists, we propose $TopScore_c$ defined as: $\frac{(\text{\#of true positives} < \text{rank } c)}{\min(c, \text{positive label count})}$. This score represents the fraction of a fixed number of experiments expected to yield a positive result, normalized by the maximum number of positive results possible. In this paper we present results for $TopScore_{10}$, $TopScore_{100}$, and $TopScore_{1000}$ for mouse, and $TopScore_{10,}$ $TopScore_{50}$, and $TopScore_{200}$ for yeast, providing insight into the usefulness of computational predictions at three different scales of experimental testing.

It is also important to note that the choice of *c* requires a certain amount of domain knowledge. For example, if there are only two true positives in the data, a $TopScore_{100}$ will show as 100% for many orderings of the first 100 examples, some of which are clearly preferable to others. If the number of true positives is entirely unknown, several values of *c* with different magnitudes should be chosen, as we have done for mouse and for yeast.

Many authors prefer the AUC_{ROC} measure when comparing algorithms, as it provides a global view of the rankings of all labels. For the protein function prediction problem, however, the skew of the dataset is generally large, and so the AUC_{ROC} score loses objective value. As seen in figure 2.2.b, a relatively poor-performing classifier can receive a very high AUC_{ROC} score, simply because the large number of true negatives implies that the algorithm “could have been much worse”. In such a case, the AUC_{PR} score can be more informative for an experimentalist, as it describes, given a goal of discovering a certain percentage of the genes that truly have a given function, what percentage of experiments will be wasted.

AUC_{PR} is not without faults, however, as the non-linearity of score can cause confusion when averaging the performance of a classifier over several different functional terms. Figure 2.2.c and 2.2.d illustrates such a case, where large improvement in one poor classifier is drowned out by a small decrease in performance of an excellent classifier.

Our TopScore metric preserves the interpretability of precision while alleviating some of the complications arising when averaging AUC_{PR} over multiple functions. Figure 2.2.c and 2.2.d illustrate the differences in TopScore alongside the average AUC_{PR}, showing that TopScore correctly captures the average improvement in classification.

2.4.5 Algorithm Component Exploration

Uncovering which component of our algorithm is responsible for which performance changes is a challenging undertaking, as many of our algorithmic changes are interlinked. For example, our choice of negative examples affects both the bias value of selected genes, but also the network combination algorithm. Additionally our bias calculation can also produce genes with a prior of *-1*, but which were not chosen as negative examples for the purposes of network combination (due to our restriction that a gene must have annotations in the branch of interest to be declared an official negative). We have performed additional experiments to isolate the performance contributions of each of our algorithm subcomponents, presented in sections 2.5.1, 2.5.2, and 2.5.3 of the results.

2.5 Results and Discussion

We present results for our proposed techniques based on the evaluation metrics and datasets described in section 2.4, along with analysis of the different components of our algorithm: negative example choice, network combination, and parameter tuning.

Our tuned ALBias algorithm shows clear advantages over the current GeneMANIA methods in the vast majority of evaluations, with these differences being especially striking in the novel evaluation scenarios, where prior biases play a more important role in the algorithm than in the test scenario. In the yeast proteome, our algorithm achieved a performance increase of 11-26 percentage points in every metric in the novel scenario, while in the mouse proteome we improved all evaluation scores by 2-6 percentage points in the novel scenario.

2.5.1 Negative Example Choice

In order to investigate the impact of our novel negative example choice, we evaluate the SW network combination algorithm with no label bias method, using three different negative example methods: the sibling negative examples, setting all non-positive genes with Gene Ontology (GO) annotations to negative examples, and our new negative example approach. As shown in Table 2.1, our negative example choice outperforms previous choices in all three full-organism evaluations, sometimes even approaching the performance of our full SWSN with ALBias and tuned parameters algorithm, indicating that our choice of negative examples is responsible for a significant part of our algorithm's final performance.

On the yeast gold standard, even though our algorithm decreases the number of validated true positives that are misclassified as negatives from 110 to 6, our negative example choice results in lower evaluation metrics than the AllNeg selection. We attribute the counterintuitive decrease in predictive performance when using ALBNeg in this setting, to the fact that the particular term MOB is specific enough to have small prevalence in the genome (only 5.3% of yeast genes possess this function), yet it is common enough that many genes have shared an annotation with it, resulting in our algorithm only selecting 691 negative examples. Thus AllNeg yields high precision by virtue of having so many more negative examples, whereby it avoids false positives, while the rarity of the term means that there are not many true positives, and thus the mislabeling of true positives is outweighed by the decrease in predicted false positives.

Despite the success of our method in increasing performance in most evaluations, and reducing the instances of mislabeling validated true positives as negatives, in some GO terms this mislabeling still occurs. Accordingly, we believe there is more potential for refined methods that correctly define high-confidence negative examples, and that these methods will have significant impact in the performance of machine learning algorithms. Indeed, we hypothesize that part of the performance gain demonstrated by the earlier HLBias algorithm, was due to the fact that the authors adjusted the labels for all non-positive genes, effectively turning any gene without a label in an ancestral term of the function in question into a negative example.

Algorithm	**AUC$_{ROC}$**	**AUC$_{PR}$**	**TS$_{10}$**	**TS$^*_{100}$**	**TS$^*_{1000}$**
Mouse Novel					
SibNeg	0.7347	0.3236	0.4103	0.5342	0.7411
AllNeg	0.8155	0.3420	0.4318	0.5783	0.8354
ALBNeg	0.8366	0.3447	0.4314	0.5793	0.8705
Mouse Test					
SibNeg	0.8573	0.5019	0.6136	0.7622	0.8725
AllNeg	0.9232	0.5168	0.6207	0.7994	0.9530
ALBNeg	0.9330	0.5171	0.6160	0.8014	0.9745
Yeast Novel					
SibNeg	0.7566	0.3090	0.3674	0.6014	0.8094
AllNeg	0.7563	0.2865	0.3299	0.5284	0.8405
ALBNeg	0.8711	0.3387	0.4133	0.7127	0.9633
Yeast Gold Standard					
SibNeg	0.7936	0.3729	0.8	0.54	0.5068
AllNeg	0.8679	0.4685	1	0.74	0.4932
ALBNeg	0.8413	0.3896	0.7	0.6	0.4865

*For the yeast scenarios, TopScore$_{100}$ and TopScore$_{1000}$ are replaced by TopScore$_{50}$ and TopScore$_{200}$

Table 2.1 Performance metrics for different negative example choices: sibling negatives (SibNeg) as in Mostafavi et al. (2008), using all non-positive genes with GO annotations as

negative (AllNeg), and negative examples based on our ALBias method (ALBNeg). All algorithms were run using the SW network combination method, and the GRF label propagation algorithm of Mostafavi et al. (2008).

2.5.2 Network Combination Algorithm SWSN

To examine the effect of our network combination algorithm, SWSN, we performed a comparison with the SW network weight algorithm, using no label biases and our negative examples, yielding mixed results across evaluation scenarios and metrics. SWSN slightly outperforms SW on the mouse novel set, the two algorithms are virtually tied on the mouse test set, and SW outperforms SWSN on the yeast novel and gold standard evaluations. Yet we believe that further refinement of negative example choice will show SWSN to be a more successful method. In order to demonstrate this, we add to the comparison of the two algorithms in Table 2.2, a third algorithm (SWSNOracle) in which our negative examples are granted access to a negative oracle, namely the validation annotations, to ensure we do not select any negative examples that are demonstrated positives (there are almost certainly others amongst our negative examples that are true positives but not yet studied at the time of the collection of validation data). This results in stronger performance on the mouse novel set and yeast gold standard, but makes no difference on the mouse test or yeast novel sets, as there were no instances of negative examples that were demonstrated positives in the mouse test and only 11 in the yeast novel benchmark.

We believe this result indicates the promise of our SWSN algorithm, despite the fact that it was likely not a significant factor in the current performance increase of our algorithm as a whole. Therefore we submit SWSN as a logical extension of SW, as it utilizes the more accurate and specific negative example information now available. We hypothesize the likelihood of future performance gain from using SWSN, once even better negative example methods are uncovered.

Algorithm	**AUC$_{ROC}$**	**AUC$_{PR}$**	**TS$_{10}$**	**TS$^*_{100}$**	**TS$^*_{1000}$**
Mouse Novel					
SW	0.8366	0.3447	0.4315	0.5793	0.8705
SWSN	0.8376	0.3460	0.4396	0.5878	0.8755
SWSNOracle	0.8775	0.3491	0.4433	0.6027	0.9366
Mouse Test					
SW	0.9330	0.5171	0.6160	0.8014	0.9745
SWSN	0.9315	0.5177	0.6211	0.8041	0.9684
SWSNOracle	0.9315	0.5177	0.6211	0.8041	0.9684
Yeast Novel					
SW	0.8711	0.3387	0.4133	0.7127	0.9633
SWSN	0.8632	0.3139	0.3796	0.6294	0.9649
SWSNOracle	0.8636	0.3139	0.3796	0.6294	0.9656
Yeast Gold Standard					
SW	0.8413	0.3896	0.7	0.6	0.4865
SWSN	0.8315	0.3729	0.7	0.52	0.5135
SWSNOracle	0.8569	0.3871	0.7	0.54	0.5270

*For the yeast scenarios, TopScore$_{100}$ and TopScore$_{1000}$ are replaced by TopScore$_{50}$ and TopScore$_{200}$

Table 2.2. Performance metrics for network combination algorithms: Simultaneous weights (SW) from Mostafavi and Morris (2010), our own SWSN algorithm, and SWSN with a negtive

oracle (SWSNOracle). All algorithms were run using the GRF label propagation method of Mostafavi et al. (2008).

2.5.3 Parameter Tuning Results

From the performance measurements presented in section 2.5.1, we see that while the tuned parameters performed significantly better than the null parameters in the mouse novel and yeast MOB benchmarks, their performance was on par with, or occasionally worse than the null guess in the mouse test and yeast novel benchmarks. We attribute the decrease in performance, primarily in the yeast novel set, to the inherent difference between the state of annotation in yeast and mouse. Our parameter-tuning algorithm was designed to re-create a learning problem where annotations are only partially known, yet in yeast, a well-studied organism, this type of learning problem was most likely not as representative of the true learning problem as it was in mouse, a less-studied organism.

In general, adapting the tuning process to be representative of the original learning problem is a more intricate problem than first anticipated, and requires further exploration. In general, the scores from several different combinations of parameters were quite similar, indicating possible fluctuation in parameter choice dependent upon the randomization in the creation of the synthetic novel tuning set.

The best parameters resulting from the tuning process in each scenario are listed in table Table 2.3, and the positive values for γ in the novel scenarios are evidence for the 2nd hypothesis put forth in section 2.3.1 of the main text, that the undiscovered occurrences of a function are dependent on its specificity and so are positively correlated with the number of annotations already observed. We also note that the μ parameter had a significant impact in all scenarios, indicating the information contained in the association network was more important than restricting genes to their prior biases.

Evaluation Scenario	λ	γ	μ
Mouse Novel	64	0.0125	0.15
Mouse Test	2	0.025	0.1
Yeast Novel (and Gold Standard)	32	0.0125	0.35

Table 2.3 Tuned parameters for each evaluation benchmark.

Further investigation shows that an observation-based guess of the parameters ($\lambda = 16$, $\gamma = -0.0125$, $\alpha = 0.4$) performs competitively with the tuned parameters. In table 2.4, the results are shown for our ALBias algorithm with naive parameters ($\lambda = 0$, $\gamma = 0$, $\alpha = 0.5$), tuned parameters (values dependent on the evaluation scenario), and the set of guessed parameters described above. The guessed parameters perform as well or better than the tuned parameters on many evaluation metrics, indicating that further work is required on the parameter tuning process. The difference in performance appears mostly separable by evaluation scenario, where it seems the tuning process works very well on the mouse novel set, but is less competitive on the mouse test and yeast novel evaluations.

In order to guarantee that the parameter tuning process provides optimal results for all evaluation settings, our synthetic novel tuning set must be more representative of the true learning task at hand. One possible solution to this problem would be to tune the parameters with a third set of actual GO annotations further back in time, so parameters would be tuned with data from year X on year X+1, then predictions made from year X+1 and evaluated with year X+2. Another possible approach would be to tailor the creation of the synthetic novel set more specifically to the proteome in question, by examining the fraction of unannotated genes, average number of annotations per gene, and other descriptive statistics.

Whatever the solution, we believe the performance gain in the mouse novel and yeast gold standard evaluation settings indicates the usefulness of the pseudocounting parameters, and that performance will increase once a more broadly applicable tuning algorithm is developed.

Algorithm	$\mathbf{AUC_{ROC}}$	$\mathbf{AUC_{PR}}$	$\mathbf{TS_{10}}$	$\mathbf{TS^*_{100}}$	$\mathbf{TS^*_{1000}}$
Mouse Novel					
NP (λ=0, γ=0, α=0.5)	0.8577	0.2773	0.3852	0.6435	0.8839
GP (λ=16, γ=-0.0125, α=0.4)	0.8604	0.3224	0.4343	0.6425	0.8887
TP (λ=64, γ=0.0125, α=0.15)	0.8616	0.3463	0.4696	0.6438	0.8956
Mouse Test					
NP (λ=0, γ=0, α=0.5)	0.9352	0.5200	0.6318	0.8157	0.9605
GP (λ=16, γ=-0.0125, α=0.4)	0.9352	0.5184	0.6377	0.8129	0.9604
TP (λ=2, γ=0.025, α=0.1)	0.9389	0.5023	0.6269	0.8222	0.9723
Yeast Novel					
NP (λ=0, γ=0, α=0.5)	0.9102	0.4917	0.6066	0.8536	0.9425
GP (λ=16, γ=-0.0125, α=0.4)	0.9122	0.4897	0.6452	0.8512	0.9432
TP (λ=32, γ=0.0125, α=0.35)	0.9000	0.4714	0.6232	0.8126	0.9405

*For the yeast novel scenario, TopScore_{100} and TopScore_{1000} are replaced by TopScore_{50} and TopScore_{200}

Table 2.4. Performance metrics for Naive Parameters (NP), Guessed Parameters (GP) and Tuned Parameters (TP).

2.5.4 Prediction Evaluations

We present the performance, evaluated by AUC_{ROC}, AUC_{PR}, and TopScore (TS) metrics, of five algorithms: the original MouseFunc GeneMANIA algorithm, the SW GeneMANIA algorithm presented in Mostafavi and Morris (2010) using sibling negative examples, the SW algorithm combined with the HLBias algorithm of Mostafavi and Morris (2009), and two versions of our algorithm, SWSN with ALBias and naive parameters ($\lambda = 0, \gamma = 0, \alpha = 0.5$), and SWSN with ALBias and tuned parameters. Results are averaged over all terms, with an analysis of results broken down by function specificity available in Figure 2.4.a.

In the novel scenario for MouseFunc, our algorithms show a strong increase in performance across all metrics, especially our version with tuned parameters, as seen in Figure 2.3.a. We see here a large difference in performance between ALBias with tuned parameters and ALBias with naive parameters, indicating that some of our algorithmic performance increase in mouse is due to the ability of our parametric pseudocounting procedure to prevent undue bias influence from understudied GO terms in mouse.

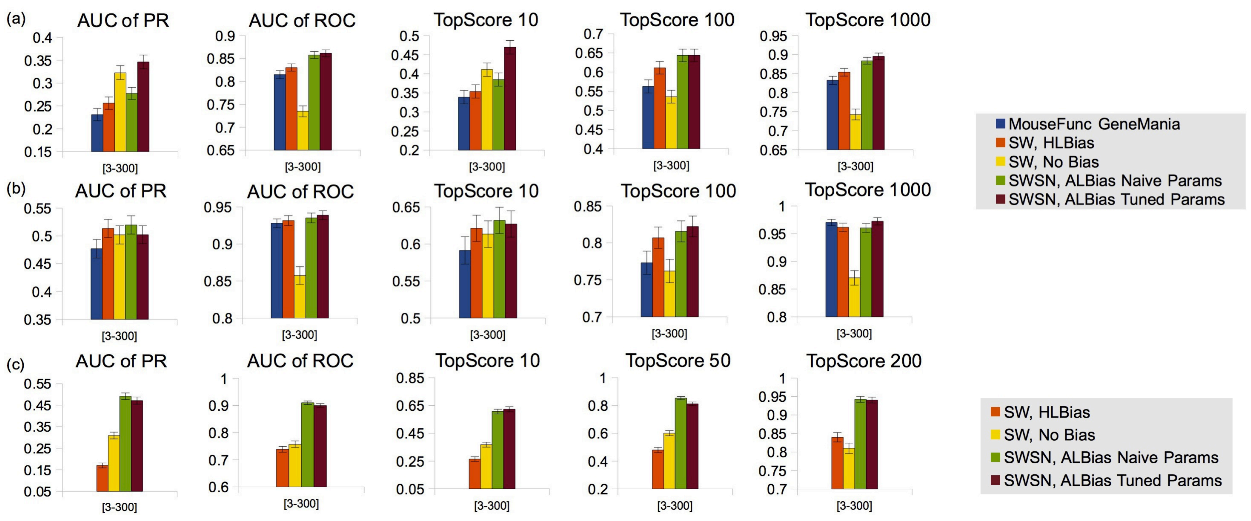


Figure 2.3 Performance metrics in (a) the novel scenario in mouse (488 functions, 1954 genes), (b) the test scenario in mouse (442 functions, 1718 genes), and (c) the novel scenario in yeast (511 functions, 342 genes). Metrics are averaged over all GO functions (each with between 3 and 300 counts per genome), and error bars are one standard deviation of the error in the

For the mouse test set, the difference in performance is much smaller, because the test set is stripped of all labels, thus negating a key advantage of ALBias (see Figure 2.3.b for mouse test results). Yet we still see a performance increase from our algorithm across most metrics, due to better biases for genes sharing edges with test genes.

In the yeast novel set we compare all algorithms except the original MouseFunc GeneMANIA algorithm, and observe a striking performance advantage of our algorithms across all evaluation metrics (See Figure 2.3.c). Further analysis indicates that much of this performance gain is due to our algorithm's incorporation of information from all branches of GO into the label bias calculation. Examining an example GO term, "DNA packaging", where our algorithm boosted performance in AUC_{ROC} from 0.722 to 0.989 and AUC_{PR} from 0.467 to 0.803, we find the primary cause to be the improvement in rankings of two true-positive genes with useful Cellular Component annotations. YBR090C-A moved from rank 102 to rank 5, due to the Cellular Component term "nuclear chromatin", which has a high joint probability with "DNA packaging", and YCL060C moved from rank 298 to 18, due to the terms "nuclear chromosome", and "chromosomal part". Further examples are provided in the Supplementary Materials.

Another example comes from term "cell cycle checkpoint" where our algorithm increased AUC_{ROC} from 0.363 to 0.974 and AUC_{PR} from 0.001 to 0.140, stemming from the improvement in rankings of two genes that had no BP data and weak affinity to positive examples in the data, but possessed useful annotations in other branches of GO. Gene YCL024W moved from a ranking of 361st to 15th, owing to its Cellular Component annotation of "cellular bud", and its Molecular Function annotations of "protein kinase activity", and "phosphotransferase activity, alcohol group as acceptor", while gene YCL060C moved from rank 101 to rank 7, thanks to CC annotations of "replication fork", "nuclear chromosome", and "chromosomal part".

A third example is the term "monosaccharide metabolic process", which showed improvement in AUC_{ROC} from 0.713 to 1.0 and AUC_{PR} from 0.514 to 1.0 from the application of our algorithm. Gene YCL040W moved from rank 256 to rank 1, owing to the label bias generated by the MF term "carbohydrate kinase activity". This term also improved the label bias for gene YCR036W, elevating it from a ranking of 54th to 3rd.

A final example lies in the term "M phase" (proteins involved in nuclear division and cytokinesis), where our algorithm increased AUC_{ROC} from 0.769 to 0.892 and AUC_{PR} from 0.408 to 0.603. We find the primary cause to be the true positive gene YHR079C-B, which moved from a ranking of 75th to 1st, despite having no Biological Process annotations in the data, thanks to its Cellular Component annotation of "condensed nuclear chromosome", which has a high joint probability with "M phase".

Many more examples exist of the contribution, primarily from Cellular Component terms, of annotations from other branches of the Gene Ontology improving the biases of genes in the prediction tasks where our algorithm improved performance the most. It is not immediately apparent why this effect was so pronounced in yeast, while other algorithmic changes seemed to be more useful in mouse, but it is clear that there is useful information to be gleaned across the different branches of the GO hierarchy when computing prior biases for function prediction.

Lastly on the yeast MOB gold standard (results in Table 2.5), we see strong performance from our tuned SWSN ALBias algorithm, which achieved significantly higher AUC_{ROC} and

TopScore$_{200}$ scores, but also from the SW, HLBias algorithm, which achieved the highest TopScore$_{10}$ and TopScore$_{50}$ scores, as well as a marginally higher AUC$_{PR}$ score. Thus our algorithm provided a better global ranking of true positives, while the current GeneMANIA algorithm ranked the top true predictions more highly.

Algorithm	AUC$_{ROC}$	AUC$_{PR}$	TS$_{10}$	TS$_{50}$	TS$_{200}$
SW, HLBias	0.8679	0.4685	1.0	0.74	0.4932
SW, No Bias	0.7908	0.3729	0.8	0.54	0.5068
SWSN, Naive Params	0.8842	0.4076	0.8	0.56	0.5068
SWSN, Tuned Params	0.9032	0.4634	0.7	0.70	0.5608

Table 2.5. Performance metrics on the yeast gold standard (GO:0007005) with experimental data from Huttenhower et al. (2009), comprised of 148 positive examples in 2815 genes. For both algorithms utilizing the SW network combination algorithm, negative examples were chosen according to the sibling technique discussed in section 2.3.2

Figure 2.4 presents the same results as Figure 2.3, but breaks down the results by the specificity of the function, using the same specificity buckets as the MouseFunc competition.

We see a strong performance increase for our SWSN, ALBias algorithm in the mouse novel evaluation setting, evenly spread across both specific and non-specific GO terms, as can been seen in Figure 2.4.a. For the mouse test set, performance was strongest for the most specific terms, but suffered somewhat in more general terms when compared to the current GeneMANIA algorithm (Figure 2.4.b). In the yeast novel setting (Figure 2.4.c), there was an interesting correlation between the specificity of the function, and the performance discrepancy between the tuned and untuned versions of our algorithm. As the specificity increased, the performance of the untuned parameters widened the gap, whereas for the most general GO terms, the tuned parameters actually performed better in nearly every metric, despite performing worse in the all-term averages.

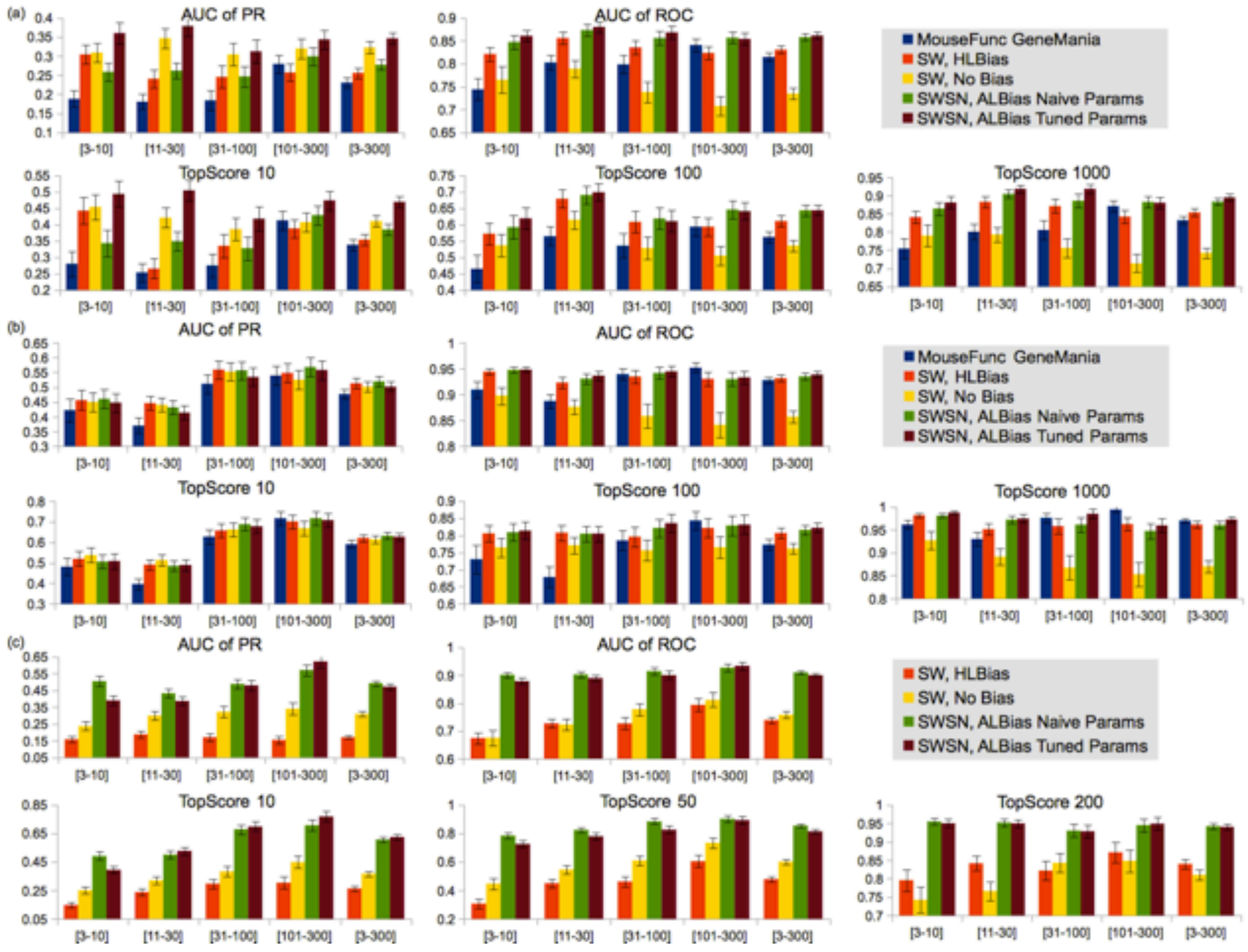


Figure 2.4 Performance metrics in (a) the novel scenario in mouse (488 functions, 1954 genes), (b) the test scenario in mouse (442 functions, 1718 genes), and (c) the novel scenario in yeast (511 functions, 342 genes). Metrics are presented for several buckets of specificity based on the number of observed occurrences in the genome in question: [3-10], [11-30], [31-100], and [101-300]. Error bars are one standard deviation of the error in the mean.

2.5.5 Computational Cost

A theoretical complexity analysis of the SBCG algorithm is not possible, but empirical testing shows a 30% reduction in the number of floating point operations (flops) required for the prediction task on original MouseFunc data, with SBCG converging to a solution with smaller residuals as well. On the yeast benchmark, the reduction in flops (where a single flop is one addition or one multiplication operation) was less, at 22%, due to the fact that the ratio between the number of genes and the number of functions to predict is much smaller (see Table 2.6). As expected, there is an observable increase in computation saved as the number of terms increases, but this is bounded by the fact that our algorithm splits the terms into subsets with a maximum size of 500. This suggests that further computation could be saved by devising a suitable strategy to deal with low condition numbers for larger sets of right-hand-side vectors. Suarjana and Law (1994) suggests that a pre-conditioner applied to the data might help reduce the number of iterations required as well.

As we have proposed a new optimization technique for computing our functional predictions, we analyze the computation cost of our algorithm compared with that employed by the original GeneMania formulation (see section 2.3.4). The computational complexity of the conjugate gradient algorithm used to solve the GRF problem in GeneMania is O(n^2) per iteration. As the algorithm must be applied to all functions, this yields a complexity of O(dn^2) per iteration where d is the total number of GO terms to be predicted. The per-iteration cost of our Successive Block Conjugate Gradient Variant is O($dn^2 + d^2n + d^3$). It is hard to imagine a case where d > n, and in fact most often n >> d, as in the original MouseFunc competition where n = 21,603 genes and d = 488 terms for the novel evaluation. In such cases the complexity of SBCG reduces to O(dn^2) as well.

Although the per-iteration complexity of both algorithms is similar, the number of iterations required is not identical, nor are the constants applied to each. Since the exact complexity of SBCGV is conditional on the size of the dependent system, and whether or not a secondary phase is required, we turn to an empirical evaluation of flops to measure algorithmic performance. Table 2.6 shows the comparison of the flops required to solve the GRF problem for each function individually with the conjugate gradient algorithm, and the flops required by the simultaneous SBCG algorithm, for different numbers of terms, as well as the final norm of the residual matrix: norm(Ax-b).

Evaluation Scenario	# of RHS	Sequential CG		SBCG	
		Flops	Error	Flops	Error
Mouse Example	10	1.8206e+11	1.4869e-08	1.5343e+11	1.1381e-08
Mouse Param. Tuning	342	5.8418e+12	6.1187e-08	4.0961e+12	5.8895e-08
Mouse Param. Tuning	1420†	2.4178e+13	1.1884e-07	1.6844e+13	9.1782e-08
Yeast Param. Tuning	426	1.9368e+11	3.3293e-08	1.5216e+11	3.7322e-08
Yeast Param. Tuning	1704‡	7.9071e+11	8.8582e-08	6.1603e+11	5.5898e-08

† split into subsets of 474, 474, and 472. ‡ split into 4 subsets of 426.

Table 2.6. Floating point operations (flops) and error (norm of the residual) results for the SBCG algorithm, and Conjugate Gradient (CG) algorithm applied sequentially.

2.6 Multi-Species Prediction

While the modifications mentioned above improve the accuracy of function prediction, all still operate within the same functional paradigm: predicting one function at a time, one species at a time. Since many proteins exist in very similar forms across different species (proteins that are essentially identical but separated by a speciation even are known as orthologs), it is reasonable to assume that there might be useful data contained across species lines. For example, if a protein has a known function in *M. musculus*, and that protein also has a known ortholog in *R. norvegicus*, it is extremely likely that one can correctly transfer the functional annotation. In addition to increasing the number of positive training examples, cross-species prediction also allows for the prediction of functional terms for which there are no currently known annotations in a target genome, as long as there are some annotations in a related species. This becomes especially important for annotation bacterial and archaea genomes, as the majority of these are currently sparsely annotated, and thus without cross-species prediction, only a limited number of functional terms can be predicted.

In order to achieve cross-species prediction, we first group organisms into families with likely orthologues. For the CAFA prediction challenge (described in section 5.2), we used the following species groupings:

Plants
Arabidopsis thaliana
Dictyostelium discoideum
Saccharomyces cerevisiae
Mammals
Homo sapiens
Mus musculus
Rattus norvegicus
Other Animals
Mus musculus
Xenopus laevis
Danio rerio
Drosophila melanogaster
Yeasts
Saccharomyces cerevisiea
Schizosaccharomyces pombe

Archaea
Halobacterium salinarum R1
Haloferax volcanii DS2
Ignicoccus hospitalis KIN4/I
Methanocaldococcus jannaschii DSM 2661
Pyrococcus furiosus, Sulfolobus solfataricus P2
Nitrosopumilus maritimus strain SCM1
Bacteria
Bacillus subtilis subsp. subtilis 168
E. coli K12, Helicobacter pylori ATCC 700392
Mycoplasma genitalium ATCC 33530
Pseudomonas aeruginosa PA01
Pseudomonas putida KT2440
Pseudomonas syringae pv. tomato str. DC3000
Salmonella enterica subsp. enterica serovar Choleraesuis strain SC-B67
Salmonella typhimurium
Streptococcus pneumoniae TIGR4

Table 2.7 Species groupings for multi-species function prediction.

Note that some well-studied organisms (M. musculus, S. cerevisea) are included in multiple groups, in order that their annotations might help predictive accuracy in some of the less studied species. We then utilized the InParanoid tool (Ostlund et al., 2010) to generate a homology score for all pairs of proteins within all species in the group (excluding pairs of proteins in the same species). As with our other data input types, we maintain sparsity by only using the 100 highest scores for each protein, and treating the rest as 0. Function prediction proceeds as normal, but where the **W** matrix before represented edges between the n proteins in a species, it now represents edges between the ($n_1 + n_2 + \ldots + n_h$) proteins in all species in the group. The structure of this resulting matrix is depicted in figure 2.5.

The block diagonal components of **W** come from the original network combination algorithm as applied to each species individually, while the off-diagonal components are all generated from the highest InParanoid scores as described above. Once the final association matrix is constructed, symmetry is enforced, and the usual normalization procedure is applied.

In the CAFA function prediction challenge (see section 5.2) more than half of the 27 target species came from the Archea and Bacteria domains, which besides being often sparsely annotated share a large number of homologous proteins. Utilizing our cross-species methodology, we were able to predict annotations in over 73 functional terms (across all branches of the Gene Ontology) which would have been impossible to predict in single-species prediction paradigms for the majority of the target organisms. For full CAFA results, see section 5.2.

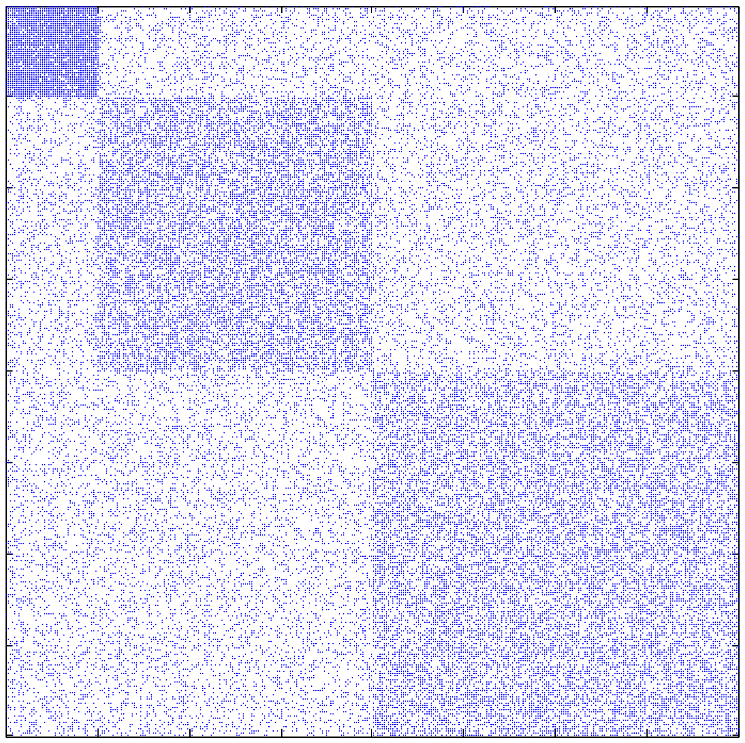


Figure 2.5 Sample block association matrix for multi-species learning with 3 species.

2.7 Conclusion

We have addressed several of the key problems facing protein function prediction efforts by proposing novel algorithms, including a method of choosing negative examples, and a parameterized Bayesian methodology for computing prior functional biases from existing annotation data. These methods, applied using the framework of the GeneMANIA function prediction algorithm, have resulted in a significant performance increases across three large benchmarks. We also introduced a new optimization methodology, which significantly decreased computational costs.

We devised a framework for tuning parameters in a synthetic novel set which added further performance gain in the novel scenario in mouse, but requires additional work to be more broadly applicable to other evaluation scenarios. Our new SWSN network combination algorithm shows even more promise in settings with more extensive negative example information. Finally, we presented a new evaluation metric designed to be easily interpretable by experimentalists, even when averaged over many function terms.

When comparing performance statistics of different algorithms, a difference of a few percentage points can mean hundreds of new true annotations when applied across all functions. For example, a 1% increase in TopScore_{10} would result in 187 new true annotations were an experimentalist to use that metric to guide experiments over the 1,874 GO MF terms in the mouse genome (at the time of MouseFunc publication). Thus we believe the algorithms presented here have the potential to guide experimentalists to a large number of fruitful assays, and are in general aligned with current biological understanding of how genes are functionally related to each other through different data types.

We have shown that our algorithm can perform function prediction through data integration and guilt-by-association with substantially more accuracy and efficiency than previously published algorithms, and provided insight into some of the inherent difficulties still facing the development and evaluation of protein function prediction algorithms.

3. Positive-Unlabeled Learning

3.1 Introduction

Despite the recent outpouring of machine learning algorithms applied to function prediction, there has been relatively little study devoted to the issue of class imbalance in function labels. This imbalance stems from the fact that the current standard set of labels for protein functions, the Gene Ontology (GO) database (Ashburner et al., 2000), rarely stores which proteins do *not* possess a function. If no annotation is present for a given gene to a particular GO term, it does not mean that such a gene is a negative example for that term, but rather that it is *either* a negative example or a positive example that has yet to be annotated. This situation arises due to experimental constraints: function assays are typically applied to single proteins and that protein function can be context dependent, making negative statements/labels quite uncertain, and leading to very few (or for most protein functions, not any) verified negative examples. This imbalance presents an obvious problem for the vast majority of machine learning techniques, which require enough examples of both the positive and negative class to train an accurate predictor. Without these labeled negative examples, authors often resort to heuristics to define the non-positive class; but mistakes stemming from these heuristics can lead to false negatives in the training set, and are detrimental to classifier performance.

The situation described above, in which the only known labels are of the positive class, is not unique to the protein function prediction (PFP) problem, but also occurs in several other domains. It has been given the name Positive-Unlabeled (PU) learning, and there has been a surge of interest lately in this particular subset of semi-supervised machine learning problems. One branch of PU algorithms attempts to learn in a one-class scenario, as has been applied to biology, specifically mRNA detection (Yousef et al. 2008). As the authors point out, however, 2-class machine algorithms often perform better when the negative class can be well defined. In another 1-step algorithm (Elkan and Noto, 2008), the authors demonstrate that if certain conditions hold, learning without explicitly knowing negative examples is possible and even more accurate then existing methods. Unfortunately, this assumption requires the probability of a true positive example being labeled to be independent of the example itself (the set of observed positive labels should be selected at random from the total set of true positives). Since GO terms are often propagated via homology methods, there is a high degree of correlation between many of the labeled positive examples, and so this assumption does not hold in our domain. Thus we focus on the majority of PU algorithms, which proceed by first predicting a set of reliable negative examples before applying a traditional machine learning classifier to the enriched data as usual. These 2-step algorithms take many forms (see Liu et el., 2003, for review of these methods), but in this work we will refer to two main subcategories: passive 2-step PU algorithms, which learn the negative examples through a separate mechanism from the classifying algorithm, and active 2-step PU algorithms, which work in conjunction with the classifier to learn the negative examples.

The main focus of PU-learning literature has been to improve text classification (Liu et el., 2003), a problem in which labeling a document's topics is time-intensive, and it is not practical to label all the topics a document does not contain. Yet the analogies to protein function are clear: proteins are rarely labeled with the functions they do NOT possess, and proteins are nearly always multi-topic, in that the annotation of a protein to a particular GO-term does not exclude the potential for several other functional classifications (we use the word "function" synonymously with "GO term", regardless of which branch of GO that term occurs in). Therefore PU algorithms are applicable to the function prediction problem, and hold great potential for improvements in machine learning algorithms applied in this context. For example, we have previously shown that more-reliable negative examples boost the predictive power of protein function prediction algorithms (Youngs et al., 2013).

We proceed by focusing directly on the first step of the PU learning task, namely generating a reliable set of negative examples for protein function and directly evaluating the quality of our negative examples, rather than their indirect effect on classifier performance. While PU learning has been applied to the biological domain before (Yousef et al., 2008; Bhardwaj et al., 2010; Zhao et al., 2008) to the best of our knowledge no study has focused on evaluating the quality of negative examples for GO functions. We examine many of the heuristics used for protein function negative examples in the past, including: designating all genes that don't have a particular label as being negative for that label (Guan et al., 2008), randomly sampling genes and assuming the probability of getting a false negative is low (often done when predicting protein-protein interactions, as in Gomez et al., 2003), and using genes with annotations in sibling terms of the term of interest as negative examples (Mostafavi and Morris, 2009; Cesa-Bianchi and Valentini, 2010). To these heuristics we add two common PU algorithms used in text classification but here adapted to PFP, the Rocchio algorithm (Rocchio 1971), and the "1-DNF" algorithm (Yu and Chang, 2002), as well as our ALBNeg algorithm (Youngs et al., 2013), and one of the few previously-published protein-negative-example-selection algorithms, the AGPS algorithm (Zhao et al., 2008). In addition, we present two new techniques: the first, Selection of Negatives through Observed Bias (SNOB), is an extension of our ALBNeg algorithm (which can itself be viewed as a generalization of the "1-DNF" PU algorithm), while the second, Negative Examples from Topic Likelihood (NETL), is based on a Latent Dirichlet Topic model of GO data.

Our algorithms, as well as competing algorithms borrowed from text classification, require only existing GO annotations to predict negative examples. As new annotations are continuously added to GO this allows testing via training on archived GO data, and examining the number of incorrectly predicted negative examples using current GO data to identify true positives that were predicted to be negative. The AGPS method utilizes additional feature data, such as Gene Expression, Protein-Protein-Interaction, etc., but can still be evaluated in the same manner as the other algorithms. We provide a case study to show how these examples can benefit the performance of other algorithms, specifically a function prediction method tested in *A. thaliana* (Puelma et al., 2012). Additionally, we demonstrate increases in function prediction accuracy when our negative examples are used, testing on human, mouse, and yeast proteins, using our earlier-published function prediction algorithm (Youngs et al., 2013). Lastly, we provide a resource, NoGO, which contains lists of high-quality negative examples for GO terms in a variety of well-studied organisms (Human, Mouse, Worm, Yeast, Rice, and Arabidopsis).

3.2 Results

3.2.1 Evaluation of Negative Example Quality

Function prediction results are biased negatively (estimations of function prediction accuracy are typically lower limits) by the fact that a positive prediction without a corresponding validation annotation might simply indicate lack of study of the gene rather than an incorrect prediction. It therefore follows that negative example validations are biased by the same effect, but positively (estimated error rates are lower bounds). Just because a gene is not annotated with the function in the validation data doesn't guarantee that it was correctly identified as a negative example. In order to attempt to rigorously evaluate potential negative example selection algorithms, we utilize the average number of false negative predictions over terms in each of the three branches of the Gene Ontology (GO).

We determine false negatives through a temporal holdout to mitigate bias (Greene and Troyanskaya, 2012), running all of our algorithms on data from the human genome obtained in Oct. 2010, and then validating with data obtained in Oct. 2012. This process involves restricting the training phase of all algorithms to data available in Oct. 2010, removing the potential for test and training data correlation that can happen during cross-validation. Any gene that was predicted as a negative example from 2010 data, which received a positive annotation in the

2012 data, is considered an error in prediction (a false negative example). For extra stringency, we consider an “Inferred by Electronic Annotation” (IEA) evidence code annotation as an indication of false negativity (even though these types of annotations are traditionally considered less reliable). For completeness, we also include an evaluation without considering IEA annotations, presented in Figures 3.9 and 3.10.

Prediction errors are calculated separately for each GO term, and then averaged together within each branch of GO. Only terms that have between 3 and 300 annotations are evaluated, so as to consider only terms specific enough to be interesting but not so specific as to have little chance of being validated, since prediction errors can be observed only if new annotations appear for the term in question in the Oct 2012 data that were not present in the Oct 2010 data.

Additionally we focus on a specific GO term in human (RNA Binding), augmenting the temporal validation with annotations from a recent high throughput screen for RNA binding proteins (Baltz et al., 2012). Lastly, we evaluate using a gold-standard set for a single GO term in the yeast genome (Huttenhower et al., 2009).

As the trivial solution (predicting no negative examples) would obviously have the lowest number of false negatives, we present results in two dimensions, where the vertical axis is average number of false negatives, and the horizontal axis is number of negative examples predicted (in this setup, the origin represents the trivial solution, while the upper right corner of the plot represents choosing all non-positive genes as negatives). Algorithms that do not have the capability to vary the number of negative examples that they predict appear as points on the performance graph, instead of lines. Because prediction errors can be evaluated only if new annotations appear during the course of the temporal holdout time period, the error rate calculated is an observed error rate, rather than the true error rate. This observed rate will vary in magnitude from GO term to GO term, as it is bounded from above by the number of new annotations. Since the magnitude of the number of false negatives in each branch is dependent on the total number of new annotations added in that branch between 2010 and 2012, the numbers cannot be compared across branches. In order to provide a reference point that is comparable across each branch, we treat the performance of random selection of negative examples as a baseline. Thus while the magnitude of the observed error rate cannot be compared across branches, the difference between an algorithm and the random baseline is comparable, both across branches and between GO-terms of differing specificity.

3.2.2 SNOB and NETL, Two New Novel Negative Selection Algorithms

Our first novel negative example selection algorithm, Selection of Negatives through Observed Bias (SNOB), is an extension of our previously published ALBNeg algorithm (Youngs et al., 2013), which selected negative examples for a function based on whether or not a gene's most specific functional annotations had ever appeared alongside that function. ALBNeg in turn can be viewed as a generalization of a popular passive 2-step PU-learning algorithm known as “1-DNF” negative example selection. This algorithm works in the context of text classification by identifying words that are enriched among the positive class, and using as negatives all unlabeled documents that do not contain any of these positive “indicator” words (Liu et al., 2003). We consider each GO term annotation as a “word” in the “document” of a protein, then apply the "1-DNF" technique to choose negative examples for a protein function by excluding proteins with GO terms that are enriched among proteins containing the function of interest.

In ALBNeg, we generalized the idea of "enrichment", by computing the empirical conditional probability of the GO function of interest, denoted *f*, given the presence of each other GO function in all three branches (Youngs et al., 2013). Proteins whose most specific annotations had non-zero conditional probabilities of appearing in a gene alongside *f* were ruled out from the potential negative set for *f*, effectively using the conditional probability as an indicator of potential positivity in the same way that the "1-DNF" algorithm uses enriched terms.

In our new algorithm (SNOB), presented here, we follow the same approach as ALBNeg, and for each GO function term *f*, compute the pairwise empirical conditional probability of seeing *f* given the presence of each other GO term. We further develop ALBNeg, i) by including IEA annotations in our calculations as well. We then obtain a score for each protein for each GO function term *f*, by averaging the conditional probabilities of all GO terms (including IEA annotations) annotated to that protein, ii) by including all GO terms in the average, not just the most specific terms, and iii) instead of choosing all proteins with a score of 0 as negatives for the function *f*, we allow the user to set a desired number *n* of negative examples, and choose the *n* proteins with the lowest scores as our negatives for *f*. See the Methods section for details of this calculation.

Our second novel algorithm, Negative Examples from Topic Likelihood (NETL), again treats proteins analogously to "documents", with the GO terms annotated to each protein serving analogously to a document's "words", but now we consider the proteins to have latent "topics" as well. These hidden topics represent the “true” function of the protein, both accounting for new functions (functions not annotated because they have to be verified/tested) as well as errors and missannotations (having a GO annotation does not guarantee that a protein actually performs the function in question due to potential errors in annotation, especially with IEA annotations). We can then apply a multi-topic inference algorithm, specifically Latent Dirichlet Allocation (Blei and Jordan, 2003), to learn the distribution of these latent topics, or “true” functions, and also learn the conditional distribution of the “words” or annotated GO terms based on those topics. Once these distributions are known, NETL selects as negatives the proteins whose latent topic distributions are as dissimilar from the positive class as possible, allowing the user to specify how many negative examples are desired.

Ideally each latent topic would represent a single GO term, but since the size of the vocabulary in our corpus is also equal to the number of GO terms, this is not feasible. Instead, we utilize the GO hierarchy to select fewer but more general topics, while ensuring coverage of the entire GO tree. Such a setup does not guarantee an intuitively interpretable relation between the latent topics and specific GO terms: topic x does not directly correlate to any one GO term, but rather is likely a combination of GO terms. Thus the calculation of the likelihood of a particular protein being a negative example for a particular GO term is infeasible, and must instead be inferred through a similarity metric (see section 3.5).

3.2.3 Previous Methods for Negative Example Prediction

In order to provide a reference for the quality of our algorithm's negative examples, we include past heuristics used for negative example selection, as well as the popular passive 2-step PU algorithms, “1-DNF” and “Rocchio”, which we have adapted to the PFP context through the GO term “word” and protein “document” mechanism described above. In the case of the Rocchio algorithm, we made an additional adjustment allowing the number of negative examples to be varied (See Methods for details). We have chosen to focus on passive 2-step PU algorithms as the performance of active 2-step methods is intertwined with the performance of the underlying classification algorithm, as well as the input feature data. A stronger classifier will produce better negative examples, as will a classifier that can use more discriminative data. This increases the difficulty of judging the relative performance of active 2-step PU algorithms, as different classifiers utilize different mechanisms and datasets. These underlying differences make it difficult to correctly attribute relative performance of negative example selection to the 2-step algorithm itself, as opposed to the quality of the classifier or underlying data. Additionally, 2-step algorithms are self-reinforcing, in that the classifier identifies as negatives those proteins which are most different from the positive class by whatever mechanism that classifier is using, which only reinforces that particular kind of discrimination when the classifier is run again with the negative examples in the second step. In general, a classifier is better served with negative examples that are actually *more* similar under the classifying mechanism, in order to force the classifier to be more discriminative. Lastly, the passive 2-step algorithms presented here function solely with GO data input, allowing for very rapid calculations and avoiding the need to gather large amounts of feature data, which can quickly become difficult for less-studied organisms.

The exception to our focus on passive 2-step algorithms is the AGPS algorithm, which is an active 2-step PU algorithm with which we make a comparison. We have included this algorithm, as it is one of the few explicit negative example selection algorithms in the protein function prediction (PFP) literature.

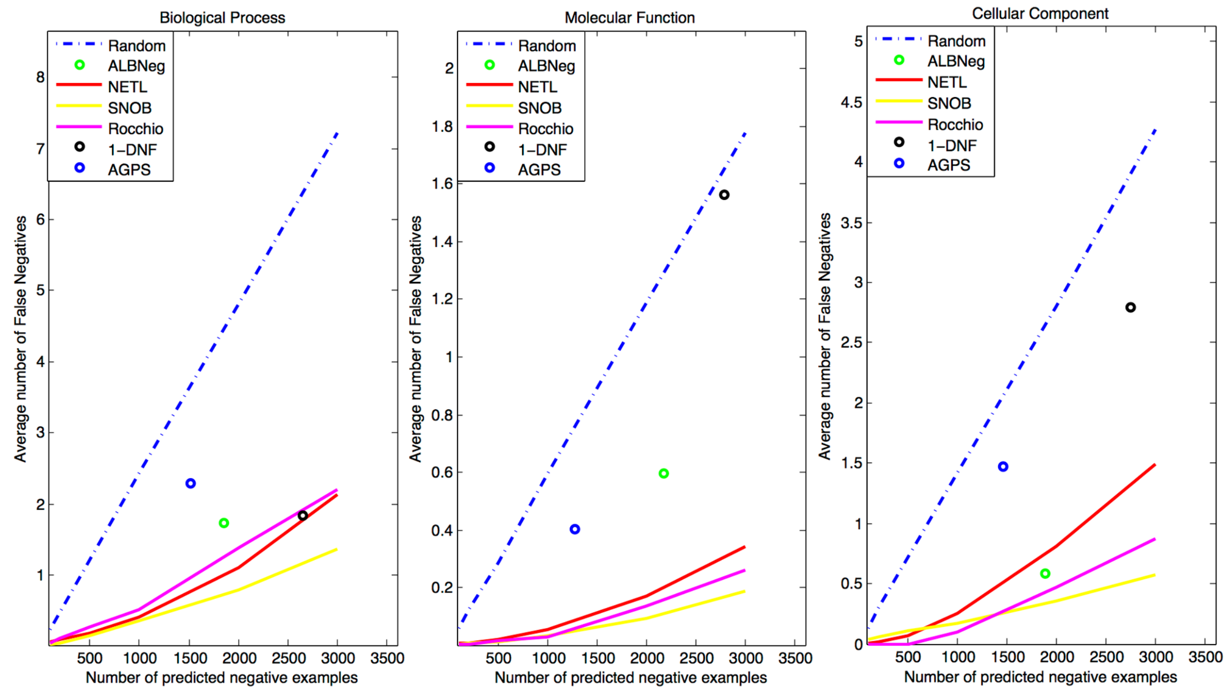


Figure 3.1 Performance measures for negative example prediction on the human genome. The number of erroneous negative example predictions is plotted as a function of the number of negative examples chosen, for each of the three branches of GO. The Rocchio, NETL, and SNOB algorithms show consistently strong performance, with SNOB achieving

the lowest error rate in each branch. The "Sibling" and "All non-positive as negative" heuristics have also been omitted, as their poor performance dramatically skewed the scale of the images.

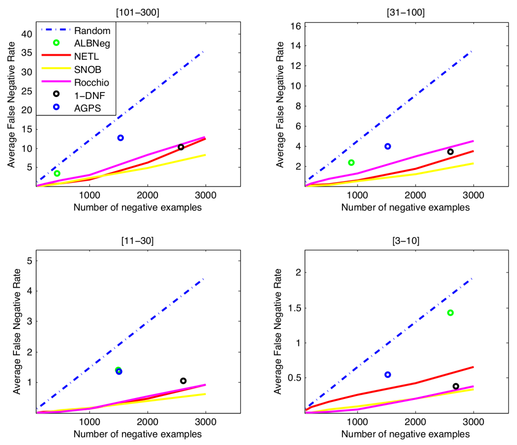
3.2.4 Performance of Negative Example Methods in *Homo sapiens*

Results for the methods tested on the human proteome are presented in Figure 3.1. Among the methods tested, all algorithms performed better than the random baseline, with the exception of the sibling algorithm, whose weakness is also confirmed in (Mostafavi and Morris, 2009). The heuristic of choosing all non-positive genes as negative also does not perform better then the baseline, as it is itself a special case of the baseline where the number of negative examples is allowed to be the size of the genome (minus the number of positive examples). The best performance was achieved by the SNOB algorithm, which achieved an equal or lower average number of false negatives than all other algorithms, heuristics, and the baseline, across all three branches. The NETL algorithm, as well as our adaptation of the Rocchio algorithm to PFP, also exhibited strong performance compared with other algorithms.

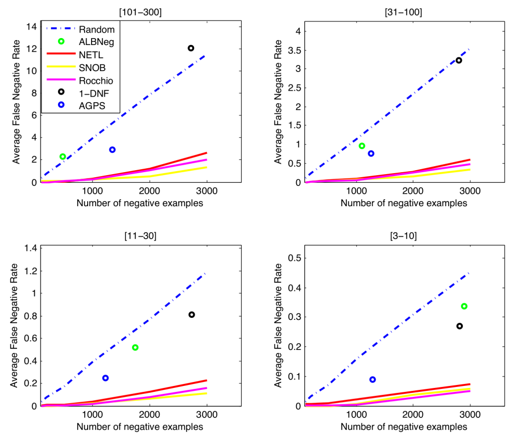
Driving the performance of SNOB was its ability to achieve significantly fewer false negative predictions for more general GO terms (terms with more annotations in the human genome). Figure 3.2 shows false negative rates broken down by the specificity of the function, demonstrating that while the Rocchio algorithm can compete with or even outperform our SNOB algorithm on the most specific terms, it is eclipsed by SNOB in the more general ones. This discrepancy among terms is most likely driven by the fact that the SNOB algorithm directly utilizes the co-occurrence of functions (See the Methods section), and thus has less information to work with for the most specific functional terms.

While not performing as well as SNOB, our previously published ALBNeg algorithm still achieves comparable or better performance than the AGPS algorithm. This comes as somewhat of a surprise, as AGPS has the benefit of access to a wealth of biological data beyond the GO information utilized by our algorithms, and much of that data post-dates the training GO annotations, providing unfair bias due to the correlation of many data types with GO annotations. However, with that additional data comes additional noise, and we recognize that the AGPS algorithm might be able to improve upon its performance with additional parameter tuning and feature selection among the data inputs.

a.


[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

b.


[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

c.

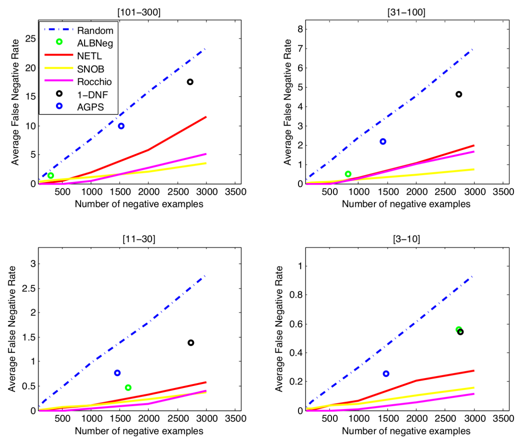

[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

Figure 3.2 Specificity-segmented performance. Performance of negative example selection algorithms broken down by specificity for **a.** Biological process, **b.** Molecular Function and **c.** Cellular component. Specificity is defined by the number of annotations present for a GO term in the human genome training data, split into buckets of size: 101-300, 31-100, 11-30, and 3-10.

The results presented in Figure 3.1 represent the average of a large number of individual evaluations, each with an error rate whose magnitude can vary largely depending upon the specificity of the term. We encourage the reader to examine Figure 3.2, which presents the same results but broken down by specificity, reducing the information lost by averaging. These results agree with those in 3.1. To further substantiate our evaluation, we focused on one particular molecular function term: GO:0003723 RNA Binding, presented in 3.3. We augmented the temporal holdout validation data with additional annotation not yet present in GO, but which have been experimentally verified in (Baltz et al., 2012) via a large-scale genomics experiment designed to detect mRNA binding proteins genome-wide. These additional annotations significantly increase the number of potential false negative examples, allowing for greater discrimination between algorithms. Continuing in the same patterns as the entire human genome evaluation, the NETL, SNOB, and Rocchio algorithms perform similarly, and significantly better than the random baseline, with SNOB edging out the other two algorithms for larger numbers of negative predictions. Both NETL and Rocchio, however, maintain a zero false negative rate for a larger number of predicted negative examples than SNOB. AGPS and ALBNeg do well, but only provide a small number of negative examples, and both predict one false negative while NETL and Rocchio achieve zero errors at the same number of negative examples. The "1-DNF" algorithm performs very poorly on this term.

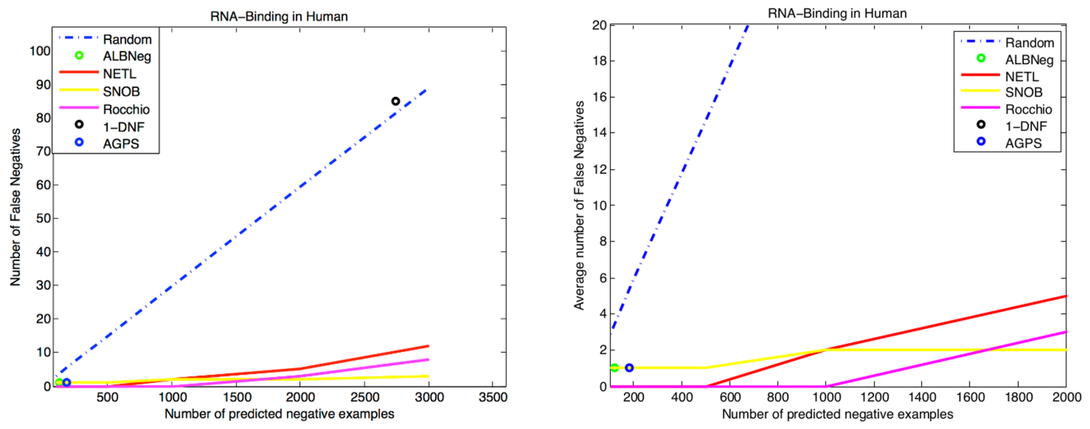


Figure 3.3 Performance measures for RNA Binding. Performance of the competing algorithms on a specific GO term: GO:0003723 RNA binding, with validation data augmented by annotations taken from (Baltz et al., 2012). The left panel shows the complete results, while the right is a scaled to see the differences between algorithms near the origin. The SNOB algorithm achieves the fewest false negatives for large numbers of negative examples, while the Rocchio and NETL algorithms maintain a zero false negative rate for a greater number of negative examples.

3.2.5 Golden Set Evaluation in *S. cerevisiae*: Mitochondrial Organisation

In order to further explore the potential biases in the evaluation of negative example selection methods, we include evaluation on a gold-standard set of annotations in yeast, obtained from (Huttenhower et al., 2009). This golden set, for the biological process term GO:0007005 Mitochondrial Organization, represents an exhaustively verified set of annotations, such that all positive and negative occurrences of this GO term are known across the entire yeast genome. Because the number of true positives and negatives is known, this GO term in yeast allows us to utilize cross-validation on the data to calculate a Receiver-Operator Characteristic (ROC) curve or point for each algorithm. While cross-validation is problematic in the evaluation of function-prediction in general, due to the interconnectedness of GO and many types of feature data which introduces large positive bias into the evaluation, here we are examining and holding out only GO terms, and so such bias is mitigated.

In the yeast golden set, we see similar results (presented in Figure 3.4) as in our evaluation with human data: The SNOB algorithm is the strongest performer, followed closely by the Rocchio and NETL algorithms. The ALBNeg algorithm also preforms well, achieving zero false assertions of negative functionality with a large number of predicted negative examples (473.2 on average). The 1-DNF algorithm also achieves zero false assertions of negative functionality, but with fewer predicted negative examples (only 76.6 on average), and the AGPS method predicts fewer negative examples than ALBNeg, with a much higher number of false negatives (2.6 on average). It is also worth noting that 59 of the 4625 negative examples in the golden set had received positive annotations for GO:0007005 in the years since the golden set was formed (the annotations set is updated accordingly here).

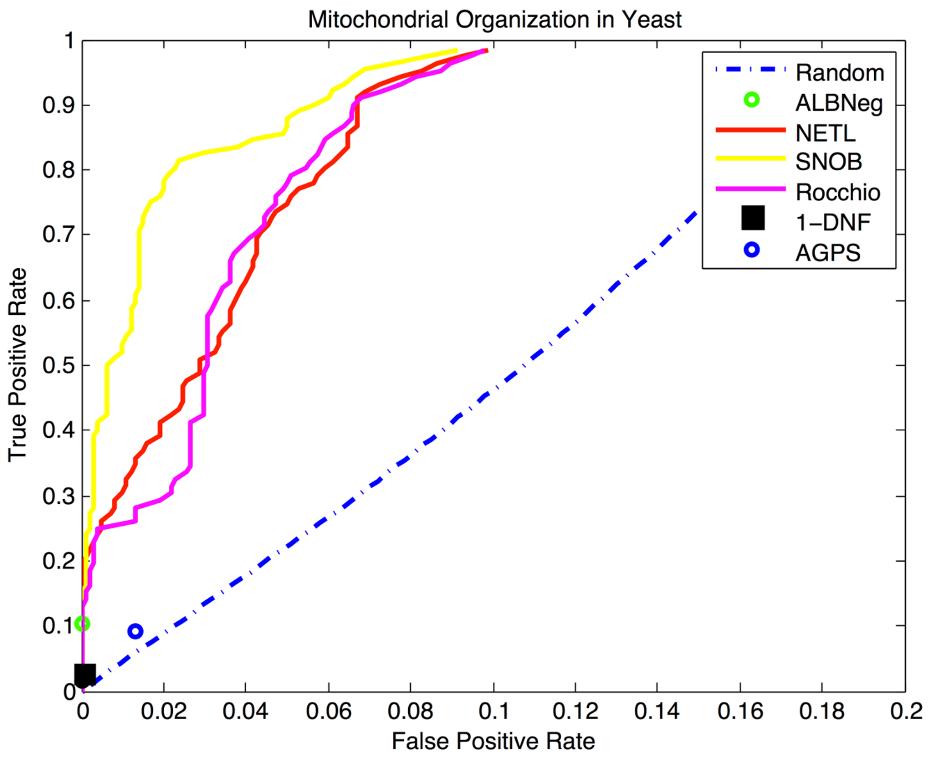


Figure 3.4 Performance measures for Mitochondrian Organization. ROC curves are depicted for each algorithm on the golden set of annotations for GO:0007005 in yeast, calculated through cross-validation. SNOB shows the highest area under the curve (AUC), followed by NETL and Rocchio, which have approximately equal AUCs.

3.2.6 Case Study: Improving function prediction in Human, Mouse, and Yeast

In order to demonstrate the importance of high quality negative examples, we use our previously published algorithm (Youngs et al., 2013) to predict functions across all three branches of GO, for human, mouse, and yeast proteins. We validate these predictions with a temporal holdout (see methods), which enables us to compute the area under the curve (AUC) for the Receiver-Operator-Characteristic (ROC) plot. We repeat this process using negative examples selected by each of the best-performing negative-example-selection methods, as well as with random negative examples to serve as a baseline. Results are presented in Figure 3.5.

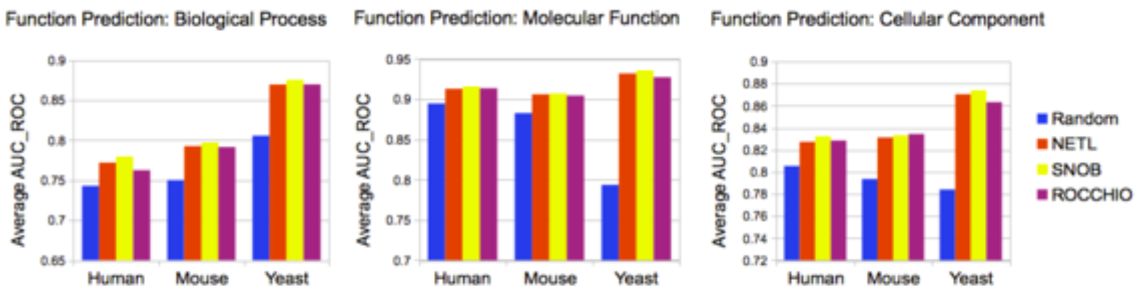


Figure 3.5 Performance measures for function prediction.

AUC_ROC measures for function prediction using the best-performing negative example selection methods, with the random negative example selector included for comparison. Performance measures are broken up by ontology branch, and represent the average AUC_ROC for all GO terms predicted in that branch.

Comparing the average AUC_ROC values of function prediction with the negative examples selected by each method, we see relative performance very similar to our earlier evaluation of negative example quality. All three of the negative-example-selection algorithms yield much stronger function prediction performance than when negative examples are selected randomly from proteins lacking the positive example. Between the three algorithms, performance is fairly similar, with function prediction utilizing the SNOB negative examples slightly outperforming the other methods.

3.2.7 Case Study: Improving function prediction in Arabidopsis Thaliana

We apply our SNOB algorithm to the work of Puelma et al. (Puelma et al., 2012), which employs discriminative local subspaces in gene expression networks to predict function in *Arabidopsis Thaliana*. We choose this work as a case study because the authors specifically mention the importance of negative examples in their work, and devise an algorithmic approach for selecting high-confidence negative examples for the 101 biological process terms they used to test their PFP method. We use their provided data to select negative examples with SNOB, generating the same number of negative examples per functional term as the author's original algorithm (a total of 313592 across all terms). Table 3.1 shows the results of our case study, demonstrating that even though our algorithm only had access to 1/3 of the data it usually requires (here the authors provided only Biological Process data, and no data from the other two branches of GO), SNOB produces significantly fewer false negatives, negative examples with greater specificity, and performs better when evaluated by the metric chosen by the authors. It is also interesting to note that even though the rate of false negatives is very small (originally only 0.6%), further reduction still produces performance gains in downstream function prediction.

Algorithm	False Negatives	Negative Frequency	Avg Enrichment P-Value
Puelma Neg	1806	71.88	39.00%
SNOB	1241	29.05	36.26%

Table 3.1 Results of our SNOB algorithm vs. the algorithm published in (Puelma et al., 2012). The "False Negatives" column shows the total number of false negatives produces by each algorithm across all 101 BP terms examined in the paper, as determined by BP data collected by the authors two years after the training data. The "Negative Frequency" column shows the average number of times any gene was selected as a negative example for different function terms, if it was selected at all (a higher number means the same proteins are showing up as negative examples across more terms). The "Avg Enrichment P-Value" column is the metric the authors used to evaluate their function predictions, with a lower value indicating better performance (see Puelma et al., 2012 for details).

3.2.8 Negative GO (NoGO) Database

We have collected negative example predictions from the SNOB, NETL, and Rocchio algorithms in an online database for use by other researchers. The database contains negative examples for Arabidopsis, Yeast, Mouse, Human, Rice, and Worm. While the NoGO database uses the most current annotations for its ranking of negative examples, we have also included false negative rates for each species in the database, obtained from temporal holdouts on older data, to allow researchers to have a reference for the quality of negative examples in that organism. We describe the quality by the area under the false negative curve, as a percentage of the area under the random baseline curve, allowing the number of negative examples to range up to 20% of the size of the genome of that organism. Results are presented in Figure 3.6.

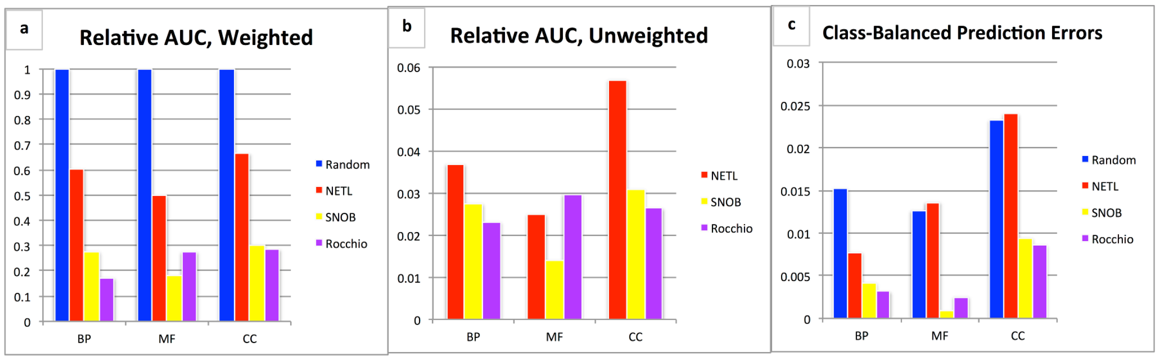
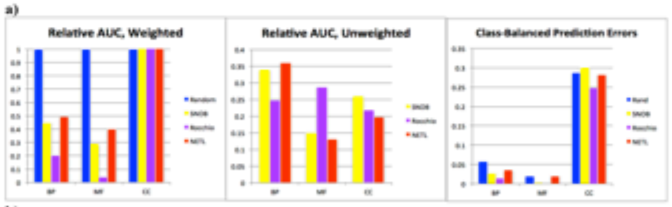


Figure 3.6 NoGO database performance statistics. Performance metrics for each algorithm in the NoGO database, averaged across all species, separated by branch of the GO ontology. A) The average area under the false negative curve, as a percentage of the area under the random baseline curve, weighted by the number of annotations in each GO term. B) The same values re-calculated so that each GO term contributes equally to the average, regardless of specificity (depicted without the random baseline as that is still 1.0 for every term but skews the scale of the plot). C) The false negative rate for each algorithm when predicting the same number of negative examples as the number of positive annotations for each GO term. Here lower numbers represent fewer errors.

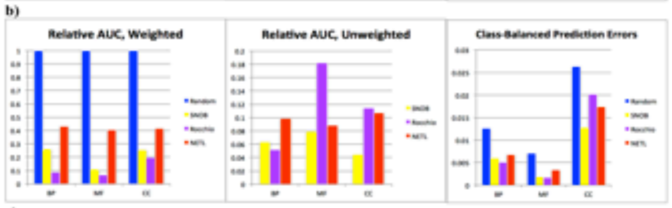
SNOB and Rocchio achieve the lowest overall errors, with the performance gap between Rocchio and NETL larger than in our other evaluations (see figure 3.7 for performance broken down by organism). The reduction of the performance gap between NETL and Rocchio in Figure 3.6b as compared to Figure 3.6a, indicates that while Rocchio performs better on more general terms, NETL's performance is on par with or better than Rocchio for the more specific GO terms (and thus a greater number of GO terms). It is also interesting to note that across all organisms, SNOB and Rocchio perform similarly on cellular component terms, SNOB has stronger performance on molecular function terms, and Rocchio performs better on biological process terms, suggesting systematic differences in the way that GO annotations relate to each other within each of the three branches.

Our Web interface to the NoGO database provides a plot for each GO function that shows the number of false negative predictions as a function of the number of negative examples chosen (Figure 3.3 is an example of such a plot, for GO:0003723). This allows researchers to make an informed decision about which algorithm to use for their specific organism, GO terms, and task. These plots also allow researchers to determine how many negative examples to use for each term (see section 3.5).

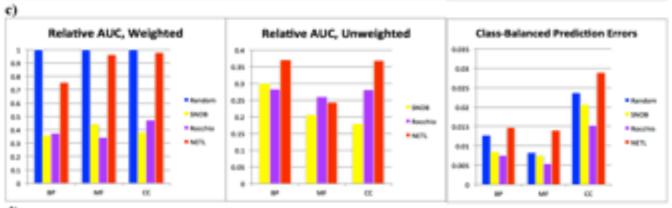
a)


Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Rand
SNOB
Rocchio
NETL

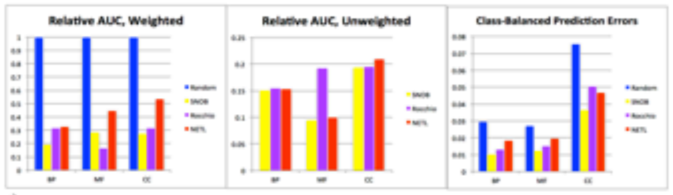
b)


Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Random
SNOB
Rocchio
NETL

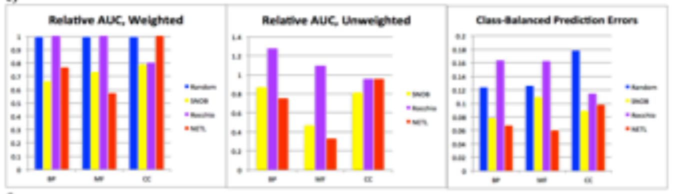
c)


Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Random
SNOB
Rocchio
NETL

d)


Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Random
SNOB
Rocchio
NETL

e)


Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Random
SNOB
Rocchio
NETL

f)

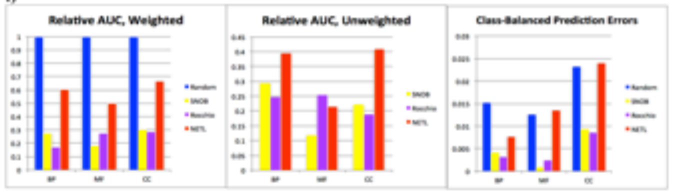

Relative AUC, Weighted
Random
SNOB
Rocchio
NETL
BP
MF
CC
Relative AUC, Unweighted
SNOB
Rocchio
NETL
Class-Balanced Prediction Errors
Random
SNOB
Rocchio
NETL

Figure 3.7. Performance metrics broken down by organism. Organism plots for a) Arabidopsis, b) Yeast, c) Mouse, d) Human, e) Rice, and f) Worm. The leftmost graph for each organism represents the average area under the false negative curve, as a percentage of the area under the random baseline curve, weighted by the number of annotations in each GO term. The central graph is the same set of values re-calculated so that each GO term contributes equally to the average, regardless of specificity. The rightmost graph depicts the false negative rate for each algorithm when predicting the same number of negative examples as the number of positive annotations for each GO term.

3.3 Discussion

We have demonstrated (using the human, yeast, and *A. Thaliana* proteomes) that the SNOB algorithm achieves significantly lower prediction errors when predicting negative examples than several previously described alternative approaches (including heuristics, techniques borrowed from PU-learning in text classification, and other negative-example prediction algorithms). These results, supported by additional literature that has explored the inter-relationships between Gene Ontology (GO) terms (Pandey et al., 2009), (King et al., 2003), indicate that despite lacking a significant number of negative annotations, the GO database encodes implicit information about likely negative examples via its positive annotations. Additionally, these pairwise term implications span all three branches of GO (cellular component, biological processes and molecular function).

Despite the success of our approach, there will inevitably be cases where the information from GO alone is not enough to predict a good set of negative examples. So-called "moonlighting" proteins, for example, can have unique combinations of functions that defy conventional annotation patterns. Additionally, approaches that rely on existing GO annotations are limited to proteins that have already been studied to some extent, which in many organisms can be a relatively small proportion of the genome. For these reasons, our group is considering active methods that can incorporate additional data types (such as gene expression, protein-protein interaction, domain structure, etc.).

The algorithms presented here represent a significant improvement over the active 2-step AGPS method that has access to data outside of GO. Our SNOB algorithm achieved a lower false negative rate than any other comparison algorithm tested, significantly lower than the "1-DNF" algorithm that served as its conceptual basis. Through our case study in Arabidopsis, SNOB also demonstrated its ability to improve existing function prediction algorithms. Youngs et al. 2013 showed that even a moderate increase in the quality of negative examples has the power to improve function prediction in general, and those results are replicated here by our case study in human, mouse, and yeast. We have shown the ability of high quality negative examples to improve function prediction accuracy, again with the SNOB algorithm achieving the best results. Additionally, this case study represents a very basic use of these negative example methods, and we believe even further accuracy can be gained by more careful selection of the number of negative examples chosen for each prediction task.

Further work includes the incorporation of additional data types, and potentially the use of active 2-step PU methods. Another potentially fruitful avenue is the explicit incorporation of the GO hierarchy in a negative example method. While GO annotations obey the "true path rule", meaning that every protein with an annotation *a* also implicitly has all annotations which are ancestors of *a*, negative annotations follow the inverse of this rule: a protein *p* that is a negative for *g* is also implicitly a negative for all descendants of *a.* This rule holds for the molecular function branch of GO, but is more complex in the biological process and cellular component branches, as there is more than one type of ancestry (terms may be direct descents, or connected by a "part-of" link, for example). These differences most likely account for some of the systematic performance differences of different algorithms on each branch of GO across all the organisms in the NoGO database.

These systematic performance differences across branches, combined with the fact that our GO-term specificity effects algorithms' relative performance, suggest the potential utility of ensemble methods (a combination of methods that use one of multiple algorithms depending on a GO term's specificity, placement in the tree, and desired size of the negative class). It is quite natural to think that the optimal algorithm will be quite different for predicting rare functions (functions that with only a handful of examples of per genome) and common functions (like information processing proteins that have hundreds of paralogous examples per genome). Further exploring the differences between the performance of NETL and SNOB for rare and common functions separately is likely to result in improved performance via hybrid methodologies.

In conclusion, we have presented a significant step forward in the calculation of negative examples for protein function prediction. Following the example set for negative protein-protein interactions by the Negetome database (Smialowski et al., 2010), we have made our predictions readily available for a variety of organisms. Our NoGO database also includes useful statistics to allow researchers to choose the number of desired negative examples and the likely false negative rate of those examples when used in their own experiments and algorithms.

3.4 Methods

3.4.1 Data Processing

Data for the human genome was obtained from the Gene Ontology (GO) database archive, with training annotations obtained from October 2010 and validation annotations from October 2012. The set of genes was obtained from HUGO by selecting all protein-coding gene symbols, resulting 19060 genes. GO terms for these genes were gathered by querying all official symbols for all annotations that have at least one annotated protein in the human genome, resulting in 7432 biological process terms, 2681 molecular function terms, and 997 cellular component terms. GO terms are fully propagated according to the "True Path Rule", meaning that an annotation of a protein to a particular term also implies annotations too all ancestral terms.

For the RNA Binding term example, there were 686 positive annotations (including annotations 'Inferred from Electronic Annotations') in our training data, and with an additional

157 annotations added in temporal holdout validation data. To these 157 new annotations, we added an additional 381 annotations, which were obtained from (Baltz et al., 2012), but are not yet present in GO. This raised the total of potential false negatives to 538.

For the case study in *Arabidopsis Thaliana*, all data was obtained from the supplementary materials provided by Puelma et al., 2012.

Annotation data for the GO:0007005 golden set in yeast was obtained from (Huttenhower et al., 2009), with training GO annotations obtained from the GO ontology in April 2013. The yeast annotations were taken for the same set of genes as the original positive and negative classes defined in (Huttenhower et al., 2009), comprised of 4966 unique yeast gene symbols, with annotations in 4226 biological process terms, 2231 molecular function terms, and 820 cellular component terms.

Data for the NoGO database was obtained from GO for each organism, with training data for the negative examples collected in April 2013, and training data for the validation plots collected in October 2011 and validated with the April 2013 data. The gene sets for each organism were also obtained from GO, by extracting all unique official gene symbols within that organism which had at least one annotation in any branch of GO. Table 3.2 lists the number of genes and GO terms for each organism, as well as the NCBI Taxa ID for each specific species used.

Organism	**NCBI Taxa ID**	**Genes**	**BP Terms**	**MF Terms**	**CC Terms**
Arabidopsis	3702	30266	3074	2338	577
Yeast	4932	6380	3533	2091	756
Mouse	10090	25488	9340	3284	1127
Human	9606	18851	9885	3732	1238
Rice	39947	58747	3115	1988	534
Worm	6239	16154	3074	1476	596

Table 3.2: Gene counts, Gene Ontology term counts, and NCBI_Taxa IDs for each of the organisms in the NoGO database.

3.4.2 Validation Plot Generation

In order to generate the validation plots in Figure 3.1 and Figure 3.2, we plot the average number of false negatives as a function of the number of negative examples. For algorithms that allow the specification of the size of the negative class, we sample the number of false negatives at 100, 200, 500, 1000, 2000, and 3000 negative examples. The average number of false negatives is determined using the temporal holdout, by seeing how many proteins that were designated as negative received an annotation in the function in question (including an IEA annotation). Functional terms that received no new annotations during the temporal holdout are not evaluated, nor are terms with fewer than 3 or more than 300 annotations. Plots are broken down by branch of the GO hierarchy, with each plot showing an average of the results for functions in that branch that meet the specified criteria. The plot for Figure 3.3 is identical in construction, but for one specific GO term, rather than an average over GO terms.

The plots in Figure 3.6 and Figure 3.7 are three representations of algorithmic performance on all organisms in the NoGO database, and each organism, respectively. The leftmost graph was generated by sampling the number of false negatives at negative class sizes equal to 0.1%, 0.5%, 1%, 2.5%, 5%, 10%, 15% and 20% of the size of the genome of the organism in question. This value is then turned into a single number by computing the area under the sample curve for each algorithm, and for the random baseline. These numbers are summed over all functional terms in the organism (or in the case of Figure 3.6 across all terms in all organisms), and then divided by the number obtained from the random baseline. The central graph is calculated identically, except here the area under the curve for each algorithm is divided by the random baseline area *before* being summed over all terms, meaning that each GO term contributes equally to the score, regardless of the number of annotations for that term. The rightmost graph represents the total false negative rate, over all GO terms in each branch, when predicting a number of negative examples equal to the number of positive annotations for that GO term. All false negative statics are obtained via a temporal holdout.

Note that in the plots for performance in the NoGO database, it is possible for algorithms to appear worse than the random baseline. This is due to the fact that the random baseline chooses from all possible unlabeled proteins, whereas the algorithms are constrained to only those proteins with GO annotations. Since it can often be the case that new annotations in the temporal holdout set are concentrated among proteins that are already partially annotated, the GO-restricted algorithms are penalized over the random baseline.

3.4.3 Selection of Negatives through Observed Bias (SNOB) Implementation

The Selection of Negatives through Observed Bias algorithm takes as its basis the pairwise conditional probability calculation of seeing annotation *a* given the presence of annotation *m,* which is specified for the ALBias algorithm in Youngs et al., 2013:

$\hat{p}(a \mid m) = {n^{+}_{ma}}/{n^{+}_{m}}$, where n^{+}_{ma} is the number of genes that *m* appears alongside *g* in the dataset, and n^{+}_{m} is the total number of genes annotated with *m* in the dataset. As mentioned in the results, SNOB removes the restriction that the score is calculated from leaf annotations only, or that a protein must have an annotation in the same branch as the GO term in question to be chosen as a negative. In addition, all annotations are utilized, including IEA annotations. The score vector $\vec{\sigma}_a$, which holds the scores for all genes as potential negative examples for a given GO function *a,* is calculated as the average of the conditional probabilities of all other annotations in each gene, which is efficiently calculable as: $\vec{\sigma}_a = \mathbf{W}^{-1}\mathbf{AP}$, where **A** is the annotation matrix of the dataset, with each row representing a gene and each column a GO term, **W** is the diagonal matrix with $\mathbf{W}_{\mathrm{ii}}$ equal to the total number of annotations for protein i, and **P** is the conditional probability matrix with $\mathbf{P}(m,a) = \hat{p}(a \mid m)$. These scores are then ranked to produce a list of negative examples, with the lower scores indicating higher probability that a particular protein is a negative example for the GO term in question.

3.4.4 Negative Examples from Topic Likelihood (NETL) Implementation

For the Negative Example from Topic Likelihood algorithm, we again formulate a protein as a document, with GO annotations (including IEA annotations) from all three branches as the words in that document. We then run Latent Dirichlet Allocation (Code obtained from David Blei's "lda-c" package) on the document corpus to identify the parameters of the Dirichlet topic distribution, and perform inference on each document to obtain the posterior topic distribution given the GO terms present in that protein (See (Blei and Jordan, 2003) for the details of LDA). Ideally, we would set the number of latent topics t equal to the number of GO terms m, but this choice yields infinite perplexity in the corpus, as the number of unique words $w = m$ as well. In order to achieve $w >> t$, to increase the quality of the learned topics, yet also to preserve coverage of all GO terms, we set the number of topics for each organism equal to the total number of annotated direct descendants of the root ontology terms. For example, in our Human validation data, the biological process node has 27 direct descendants with annotations in the data, the molecular function node has 14 direct descendants, and the cellular component node has 10, for a total of 51 latent topics. By invoking the inverse of the true path rule, whereby negative examples are propagated downwards through the GO graph, this approach guarantees coverage of all GO terms for the purposes of negative example selection.

Since LDA discovers latent topics, which are not predefined before the algorithm is run, it is not immediately obvious which learned topic corresponds to which GO term. Indeed despite our efforts to ensure coverage of every GO term directly descended from a root node, it is not necessarily the case that the correspondence between the topics and the selected GO terms are 1-

1. Instead it is possible, even likely, that some combinations of topics/GO terms relate to each other, making exact inference of the probability that a given protein possesses a given GO term difficult under the LDA model. To overcome this problem, we chose to represent the positive class with the average of the Dirichlet posterior vectors for all proteins annotated to the function in question (including IEA annotations). Then for each unlabeled protein *u*, we calculate a Distributional-Overlap Score (DOS) representing the similarity of topics distributions between *u* and the positive class average topic distribution. This score can be viewed as a symmetric simplification of the Kullback-Leibler Divergence metric, and is calculated simply as $DOS({}_i\vec{\alpha}, {}_j\vec{\alpha}) = \sum_t \min({}_i\alpha_t, {}_j\alpha_t)$, where ${}_i\alpha$ and ${}_i\alpha$ are two Dirichlet posterior parameter vectors (since each posterior vector sums to 1, the DOS score is also bounded by [0,1]). The unlabeled proteins are then ranked according to this score, with the lowest DOS values indicating the most likely negative proteins, as these are proteins which are least likely to share topics with the positive class of proteins.

3.4.5 Random Baseline Implementation

In order to calculate the random baseline, we consider the positive class to be all proteins with an annotation in the function of interest (including an IEA annotation), and all other proteins to be the unlabeled class. We sample uniformly at random without replacement from those unlabeled proteins to pick negative examples, allowing the user to specify the desired size of the negative class. In order to reduce noise from this stochastic operation, we calculate the baseline 100 times for each branch of GO, and then display the average of those 100 calculations.

3.4.6 Rocchio Implementation

In order to adapt the Rocchio algorithm to protein function, we follow the pseudocode in (Rocchio, 1971), treating the set of GO terms across all three branches as our lexicography, each protein as a document, and the annotations of that protein as a word. This formulation allows the computation of the *tf-idf* vectors required by the algorithm, and for each function we treat the positive class as all proteins with an annotation in that function (including IEA annotations), and the rest of the proteins as the unlabeled class. The algorithm then builds a representative vector for the positive and unlabeled class, and computes the cosine similarity of the *tf-idf* vector for each unlabeled protein with each of the representative vectors. Where the traditional algorithm would assign as negative examples all proteins whose similarity to the unlabeled class vector is greater than to the positive class vector, we assign a score to each protein, defined as: UnlabeledSimilarity – PositiveSimilarity. This allows us to rank the proteins in terms of confidence of their negativity, with the highest-scoring proteins as the most likely to be negative examples.

3.4.7 1-DNF Implementation

For the 1-DNF algorithm, we again formulate proteins as documents and GO terms across all three branches as words. We proceed according to the pseudocode laid out in (Liu et al., 2003), utilizing as the positive class all proteins with an annotation in the function of interest (including IEA annotations). Other GO terms that appear more frequently in the positive set than the unlabeled set are considered our "enriched" words, and negative examples are all proteins that are not in the positive class and do not contain any of these enriched words. As there is no immediately obvious way to translate this decision into a score, we only implemented this algorithm for one choice of the number of negative examples, rather than thresholding it to allow the user to specify the desired size of the negative class.

3.4.8 AGPS Implementation

Code for the AGPS algorithm was generously provided by the authors of (Zhao et al., 2008). AGPS requires features to operate, which we obtained through the similarity networks provided by the Genemania server (Wade-Farely et al., 2010). Each of these networks (235 networks for human, 297 for yeast) represents similarity between pairs of genes according to a particular datatype. For human data it was necessary to translate the networks from being specified by ENSEMBL ids to gene symbols by using the HUGO lookup for gene symbol and ENSEMBL pairs. For both yeast and human, we performed a simple linear combination of all of the networks, where each component network and the final network was normalized according to the scheme: $N' = D^{-\frac{1}{2}} N D^{-\frac{1}{2}}$, where D is the diagonal row sum matrix of W. Once the final network was obtained (a 19060x19060 matrix for human, 4966x4966 for yeast), we applied Principal Component Analysis to reduce the feature size to a 19060x200 matrix and a 4966x200 matrix, which were the input feature sets for AGPS for each organism, respectively. We ran the algorithm provided by the authors using all of the default constants provided, but as described in the author's text, ran cross-validation for each term and only used negative examples that were chosen in the majority of the cross validation runs. We choose to segment data into 5 cross-validation segments.

AGPS was only validated on functional terms with at least 85 annotations (the reliance of the method on cross-validation increases the number of necessary positive examples for a meaningful result). The lengthy runtime of the algorithm also restricted our application of the method to function terms with more than 85 annotations. To allow for a fair comparison to other methods we utilized the inverse of the true path rule, and for GO functions with fewer than 85 annotations in the human genome, we set the negative examples as the union of all of the negative examples of all parent terms of that GO term.

3.4.9 Sibling Heuristic Implementation

For the heuristic that chooses siblings as negatives for a function, we follow the specification laid out in (Cesa-Bianchi and Valentini, 2010), whereby a protein is a negative for a function if it is annotated to the parent of that function, but not to the function itself. This includes proteins annotated to sibling terms, as well as those annotated to the parent but to none of the children of that parent. Because some function terms will have no proteins that satisfy these requirements, we revert in this case to the strategy of choosing all non-positive proteins as negative, where the positive class is all proteins with an annotation in the function in question (not including annotations that were ‘Inferred from Electronic Annotation’). As Mostafavi 2009 points out, the sibling approach is problematic in that many sibling terms are not mutually exclusive, but we present the technique here for completeness. Since the heuristic will produce different numbers of negative examples for different function terms, the point on the validation plot corresponding to this algorithm represents an average over different sizes of the negative class.

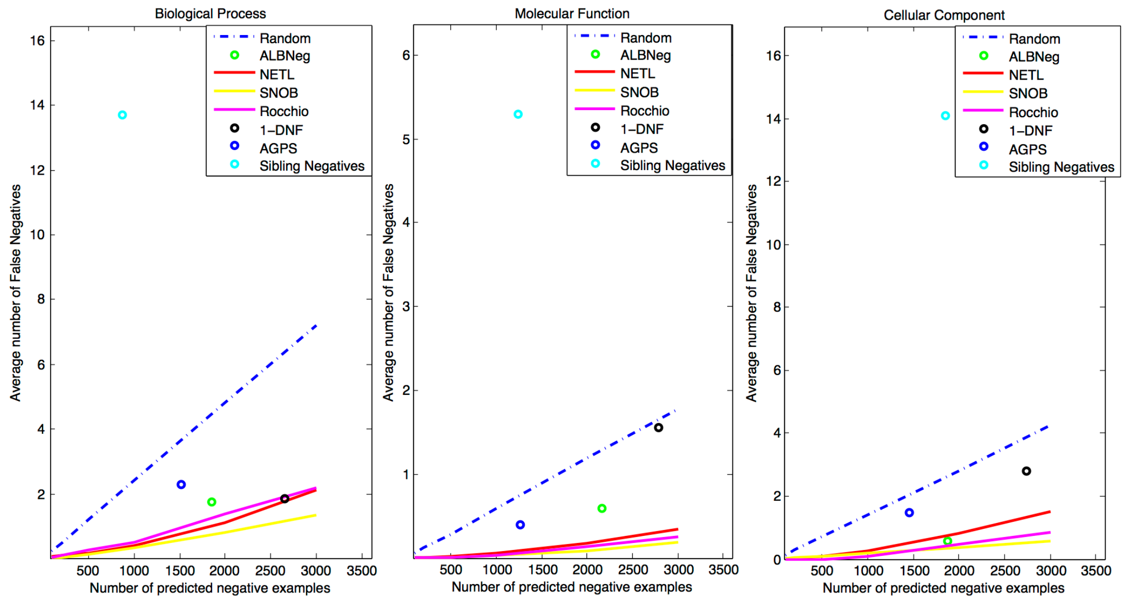


Figure 3.8 Performance measures including the sibling method. These plots are duplicates of the performance plots in Figure 3.1, but including the Sibling Negatives heuristic, to illustrate the poor performance of that heuristic.

3.4.10 Function Prediction Implementation

For function prediction, we used our previously published algorithm (Youngs et al., 2013). Training GO annotations were obtained from the GO archive in April 2013, with validation annotations obtained in December 2013. Input data included protein-protein interaction, Interpro database data (Jones et al., 2014), gene expression data, sequence similarity, and phylogenetic profiles. Predictions were made for all terms in all three branches, regardless of specificity, but validations were calculated only for those terms that received new annotations during the temporal holdout period.

For each term predicted, the number of negative examples was selected to be the maximum of the number of positive examples of that term, or 20% of the size of the genome. A further restriction capped the number of negative examples at 50% of the number of non-positive genes for the function in question. The area under the curve of the Receiver Operator Characteristic plot was calculated using the methodology presented in (Youngs et al., 2013).

3.4.11 Data Access

Negative examples are available in the NoGO database, located at: bonneaulab.bio.nyu.edu/nogo.html. Negative examples are currently available for the following species: Human, Mouse, Yeast, Rice, Arabidopsis and Worm. For each function in each organism, a ranked list of genes shows the most to least likely negative examples, available for the SNOB, NETL, and Rocchio algorithms described here. All negative examples were computed using GO data from April 2013.

Accompanying each list is a validation plot (See Figure 3.3 for a sample, GO:0003723 in Homo Sapiens), which shows the performance of SNOB against a random baseline, trained on GO data obtained from October 2012 and validated with data from April 2013. This plot gives a researcher an idea of the relative performance of the SNOB algorithm against the random reference, in order to give confidence as to the likelihood of false negatives, and also allows a researcher insight into how many negative examples to choose based on the false negative rate presented in the graph.

MATLAB code for generating negative examples from custom data will also be available from the downloads section of the NoGO database, as well as directly from: http://markula.bio.nyu.edu:8080/downloads.

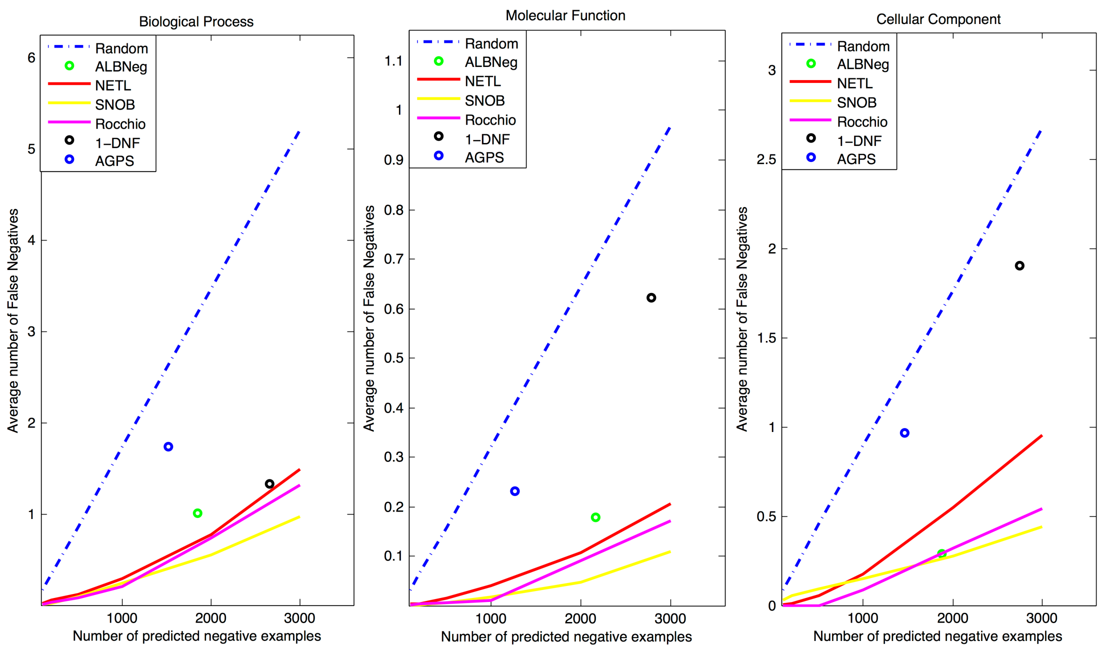
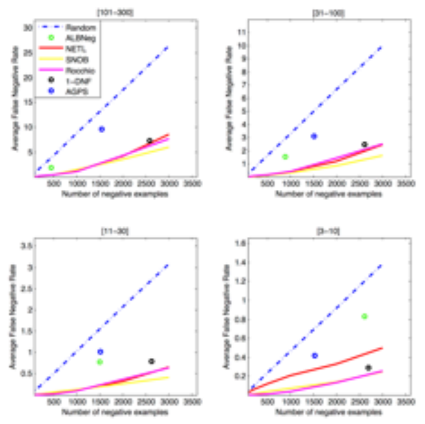
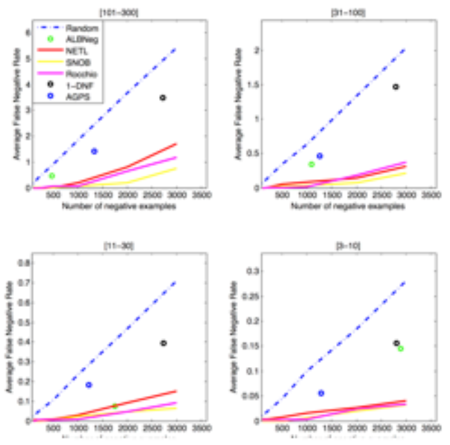


Figure 3.9 Performance measures evaluated without annotations 'Inferred by Electronic Annotation' (IEA). Performance measures for negative example prediction on the human genome, in each of the three branches of GO. These results are the similar as those presented in Figure 3.1, with the difference being that here error rates are calculated using only curated GO annotations, and ignoring IEA annotations.

a.


[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

b.


[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

c.

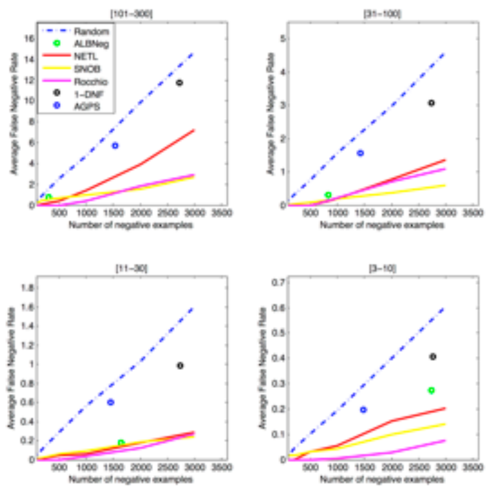

[101–300]
[31–100]
[11–30]
[3–10]
Random
ALBNeg
NETL
SNOB
Rocchio
1–DNF
AGPS
Average False Negative Rate
Number of negative examples

Figure 3.10. Specificity-segmented performance evaluated without annotations 'Inferred from Electronic Annotation' (IEA). Performance of negative example selection algorithms broken down by specificity for **a.** Biological process, **b.** Molecular Function and **c.** Cellular component. Specificity is defined by the number of annotations present for a GO term in the human genome training data, split into buckets of size: 101-300, 31-100, 11-30, and 3-10. These results are similar to those presented in Figure 3.2, with the difference being that here error rates are calculated using only curated GO annotations, and ignoring IEA annotations.

3.5 Theoretical Concerns

3.5.1 Optimal Learning Under a Key Assumption

Taking a step back from the specific application of protein function prediction, we now examine some theoretical implications for PU-learning in general. An important work by Elkan and Noto (2008) explores a different approach to PU-learning than those described above. Namely, they show that the decision boundary of a classifier trained to differentiate between positive and unlabeled examples (rather than guessing negative examples from the set of unlabeled examples, and then training a classifier to discriminate between positive and negative) will produce predictions that obey the same rank-ordering as a "traditional" classifier that attempts to discriminate between positive and negative examples. This is an important result, as it provides a mechanism to train a model without the need to guess negative examples, which will have the same ROC_AUC value as a model trained on the positive and negative examples. Additionally, Elkan and Noto show that one can also adjust the decision threshold of their “non-traditional” classifier, such that it’s accuracy is also in-line with a “traditional” classifier. The authors prove the theoretical principles behind their results, and then offer two methodologies that take advantage of those principles, demonstrating superior performance on a real-world PU-learning problem, over some other PU-learning algorithms.

Elkan and Noto, however, note that their work relies on one crucial assumption: that the set of labeled positive examples is chosen uniformly at random from the set of all positive examples. While this assumption might appear innocuous, in many real-world PU applications, it is certainly violated. In protein function prediction, for example, annotations are very often propagated via homology to known sequences, meaning that the labeled positives of a function are not selected at random from all proteins with that function, but rather according to a bias based on sequence similarity to the first proteins for which that function was annotated. It is not difficult to imagine other problem scenarios, such as text classification, where documents are given to curators based on search queries, and thus the set of labeled positive examples are again biased rather than random.

We examine the effects that the violation of this selected-at-random assumption has on the algorithms of Elkan and Noto, both on a highly controlled synthetic dataset, as well as some real-world text classification problems. In addition, the test case provided by Elkan and Noto is one where the number of labeled positive examples far exceeds the number of remaining unlabeled positive examples, a situation which may not always appear in practice. We examine the case where the opposite is true: the number of remaining positives is much larger than the number of labeled positives, and see what effect this has on the algorithms presented in Elkan and Noto as well. We also present novel algorithms based upon our previous work with NETL and SNOB, and compare their performance in these scenarios.

3.5.2 Theoretical Framework

We adopt the theoretical framework used in Elkan and Noto (2008). Let *x* be an example with a binary label $y \in \{0,1\}$. Let *s* be a second binary label for *x*, which indicates whether the value of *y* is known. Thus *y* is an indicator of whether or not the example is "positive" or "negative", to use traditional nomenclature, while *s* is an indicator as to whether *x* is "labeled" or "unlabeled". Since only positive examples are labeled in a PU scenario, we have the following axioms:

$$s = 1 \rightarrow y = 1$$

$$s = 0 \rightarrow y = 1 \; or \; y = 0$$

$$p(s = 1|x, y = 0) = 0$$

The assumption stated above, that the labeled examples are chosen uniformly at random from the set of positive examples, can be expressed by the following:

$$p(s = 1|x, y = 1) = p(s = 1|y = 1) = c$$

So the probability that any given positive example is labeled is a constant.

Within this formalism, it would be the goal of a "traditional" machine learning classifier to classify the probability of a given example being positive. If we call such a classifier *f*(*x*), then we have:

$$f(x) = p(y = 1|x)$$

Since we do not know all the values of y, training such a classifier is difficult, but Elkan and Noto instead propose to train a classifier, $g(x)$, which instead determines the probability that an example will be labeled:

$$g(x) = p(s = 1|x)$$

Given the assumption above, Elkan and Noto show that these two classifiers can be related to each other:

$$\begin{aligned} p(s = 1|x) &= p(y = 1 \wedge s = 1|x) \\ &= p(y = 1|x)p(s = 1|y = 1, x) + p(y = 0|x)p(s = 1|y = 0, x) \\ &= p(y = 1|x)p(s = 1|y = 1, x) \\ &= p(y = 1|x)p(s = 1|y = 1) \\ &= p(y=1|x)c \\ g(x) &= cf(x) \end{aligned}$$

Where the third line follows since $p(s = 1|x, y = 0) = 0$, and the fourth line follows from the "selected at random" assumption. This result shows that $f(x)$ is an increasing function of $g(x)$, meaning that examples ranked by their $g(x)$ score will have the same ordering as examples ranked by the "traditional" classifier $f(x)$. Additionally, exact values for $f(x)$ are obtainable from values of $g(x)$, so long as a good estimate of c is attainable, which the others demonstrate is indeed the case.

3.5.3 Biased Labeling

When the process for labeling a new positive example is at all dependent on the set of currently labeled examples (such as in protein function prediction, where new positives are often identified via homology to existing positives), the “selected at random” assumption is no longer valid. The probability of a positive example being labeled is now some function of that example itself, say *h*(*x*):

$$h(x) = p(s = 1|y = 1, x)$$

The result of Elkan and Noto, (2008) now becomes:

(Equation 3.1)

$$g(x) = h(x)f(x)$$

$$f(x) = \frac{g(x)}{h(x)}$$

This new relation between *f*(*x*) and *g*(*x*) now relies on the ability to estimate *h*(x) to transform the values of the learnable “nontraditional” classifier into those of the in-estimable “traditional” classifier. Additionally, the ranking assumption no longer holds, unless there are specific conditions placed upon h(x). Namely:

$$f(x_1) \geq f(x_2) \geq f(x_3) \;\leftrightarrow\; g(x_1) \geq g(x_2) \geq g(x_3)$$

i.f.f.

$$h(x_1) \geq h(x_2) \geq h(x_3)$$

Proof:

$$let \quad f(x_1) \geq f(x_2) \geq f(x_3)$$

$$\frac{g(x_1)}{h(x_1)} \geq \frac{g(x_2)}{h(x_2)} \geq \frac{g(x_3)}{h(x_3)}$$

$$g(x_1) \geq \frac{h(x_1)}{h(x_2)} g(x_2) \quad and \quad g(x_2) \geq \frac{h(x_2)}{h(x_3)} g(x_3)$$

$$g(x_1) \geq g(x_2) \geq g(x_3) \quad is\ gauranteed\ only\ if \quad \frac{h(x_1)}{h(x_2)} \geq 1 \ and \ \frac{h(x_2)}{h(x_3)} \geq 1$$

$$which\ is\ equivalent\ to:$$

$$h(x_1) \geq h(x_2) \geq h(x_3)$$

This result states that the ranking assumption only holds if the labeling bias is identically ranked, i.e. if example *i* is more likely to be positive than example *j* then *i* is also more likely to be labeled than *j*. This is not a prohibitive restriction, as it is not difficult to imagine a scenario in which new examples are labeled according to their similarity to existing examples, but the restriction is worth noting.

3.5.4 A Novel PU-Learning Method: SNOBProb

We introduce a new PU-Learning method based upon our previous work with the SNOB algorithm for function prediction (see section 3.5.3). We generalize this algorithm into SNOBProb, which is applicable to any type of feature data (both discrete features, like protein function labels, and continuous features). In addition, SNOBProb provides calibrated probabilities of a particular example being positive or negative, allowing for a weighted training paradigm like that suggested in Elkan and Noto, (2008). In this paradigm, a duplicate example is introduced to the training set for each training example of unknown label, and a PU-algorithm is utilized to calculate the probability that a given unlabeled example might be positive or negative, with that probability then used to weight each example and its duplicate, one with a positive label and the other with a negative.

Formally, for each example *x* in the training set, we desire a function *D*, such that:

(Equation 3.2)

$$\begin{gathered} D: x \rightarrow \Re^{n} \quad s.t. \\ D(x_p) \sim \mathrm{N}(\mu_p, \sigma_p) \quad and \quad D(x_n) \sim \mathrm{N}(\mu_n, \sigma_n) \end{gathered}$$

The first condition allows us to estimate μ_p and σ_p, since we have a set of labeled positive examples in the training data, but the values for μ_n and σ_n are not, as all we have is a set of examples drawn from the distribution: $\mathrm{N}(\mu_u, \sigma_u) \ = \mathrm{N}(\mu_n, \sigma_n) + \mathrm{N}(\mu_q, \sigma_q)$. That is to say, we can estimate the parameters of the distribution on the unlabeled training examples, but the results will be a mixture of Gaussians, with greater bias away from $\mathrm{N}(\mu_n, \sigma_n)$ the more true positives are included in the unlabeled set (the greater the size of *q*).

Once a function *D* is chosen, estimates are obtained for parameters of the positive and unlabeled distributions. For each example, the probability is computed of that example belonging to *p*, or to our proxy for *n* (which is trained on *n* + *q*). This calculation can be achieved either via a closed-form solution, or by Monte Carlo sampling (depending upon the number of dimensions in D). In the one dimensional example, a Monte Carlo estimate can be obtained for example *x* by generating *n* points from the distribution $\mathrm{N}(\mu_p, \sigma_p)$, and *n* points from the distribution $\mathrm{N}(\mu_u, \sigma_u)$.

Then an estimate for the probability that *x* is in the positive class, is given by:

$$p(y = 1|x) \sim \frac{\sum_{i=1}^{n} \mathcal{I}(k_i; x)}{\sum_{i=1}^{n} \mathcal{I}\,(k_i; x) + \sum_{i=1}^{n} \mathcal{J}(l_i; x)}$$

Where k_i is the ith Monte Carlo point generated from $\mathrm{N}(\mu_p, \sigma_p)$, l_i is the ith Monte Carlo point generated from $\mathrm{N}(\mu_u, \sigma_u)$, and $\mathcal{I}(m; x)$ and $\mathcal{J}(m; x)$ are indicator functions such that:

$$\mathcal{I}(m; x) = \begin{cases} 1 \; if \; m < D(x) \\ 0 \; if \; m \; \geq D(x) \end{cases}$$

$$\mathcal{J}(m; x) = \begin{cases} 1 \; if \; m > D(x) \\ 0 \; if \; m \; \leq D(x) \end{cases}$$

This equations assumes $\mu_p > \mu_q$, but in the case that the opposite is true, an equation for $p(y = 1|x)$ follows in the same manner. Intuitively, if *D*(*x*) were so large that it was greater than all Monte Carlo points generated from both $\mathrm{N}(\mu_p, \sigma_p)$ and $\mathrm{N}(\mu_u, \sigma_u)$ the approximate probability of *x* coming from the positive class would be: $p(y = 1|x) = {}^{n}\!/_{(n + 0)} = 1$. Conversely, if *D*(*x*) were so small that it was less than all Monte Carlo points generated from both $\mathrm{N}(\mu_p, \sigma_p)$ and $\mathrm{N}(\mu_u, \sigma_u)$ the approximate probability of *x* coming from the positive class would be: $p(y = 1|x) = {}^{0}\!/_{(0 + n)} = 0$. Finally, if *D*(*x*) were larger than half of the Monte Carlo points generated from both $\mathrm{N}(\mu_p, \sigma_p)$, but also smaller than half of the Monte Carlo points generated from both $\mathrm{N}(\mu_u, \sigma_u)$, then the approximate probability of *x* coming from the positive class would be: $p(y = 1|x) = {}^{n/2}\!/_{(n/2 + n/2)}$ = 0.5.

Once the probabilities have been approximated, they can be used in the methodology described at the beginning of this subsection,, where unlabeled training examples are duplicated, with one half being given positive labels and weight = $p(y = 1|x)$, and the other half negative labels with the weight = $p(y = 0|x) = 1 - p(y = 1|x)$. Additionally, we explore another method whereby unlabeled examples are not duplicated, but rather all treated as negative examples, but with weight = $p(y = 0|x)$. We refer to the former approach as SNOBProb$_{\text{ElkNot}}$, and the latter approach as SNOBProb$_{\text{Neg}}$.

Lastly, we implement one additional method, which we refer to as SNOB$_{\text{raw}}$, that bypasses the Monte Carlo probability estimation, and simply uses the raw SNOB score (the value of $D(x)$) as a weight on the unlabeled examples, which are then given negative labels. In order to adjust the raw SNOB score into a probabilistically interpretable score, we scale by the maximum value of raw SNOB scores for the known positive examples. This yields weights in [0,1], with all positive examples given a weight of 1, unknown examples with no available data from which to compute a score given a weight of 0.5, and unknown examples with raw SNOB scores given the scaled weights mentioned above.

3.5.5 Synthetic Data Generation

In order to test the effect of bias on the PU learning scenario, we create 2-dimensional synthetic data from two Gaussian distributions. We generate positive examples from a distribution with mean: (5,10), and standard deviation: $\begin{bmatrix} 1 & 0.5 \\ 0.5 & 2 \end{bmatrix}$, and negative examples from a distribution with mean: (1,5), and standard deviation: $\begin{bmatrix} 1.5 & 0.5 \\ 0.5 & 2.5 \end{bmatrix}$. We explore scenarios with different numbers of labeled positive examples, p, unlabeled positive examples, q, and unlabeled negative examples, n. Additionally, when selecting the labeled positive examples from the set of all true positives, we either select uniformly at random, or according to a bias function: $bias(x) = \sqrt{x_1^2 + x_2^2}$. Thus examples which are farther away from the origin are selected first, and since this direction is roughly perpendicular to the decision boundary between positive and negative values, creates a difficult learning problem. In order to explore the effect of differing amount of bias, we choose some labeled examples at random, and others according to the bias function. For example, if p = 100, and we set the bias amount to 40%, then 60 of the labeled positive examples will be chosen uniformly at random from the set of all true positives, while the remaining 40 will be chosen according to the bias function described above. Figure **3.x** shows one example of synthetic data, along with scenarios with 0% and 100% bias.

We compare the results of 7 algorithms on this synthetic data: 1) The optimal classifier, trained on the true positives and true negatives (this provides an upper bound on performance on the generated data), 2) The scaled SVM proposed by Elkan and Noto, (2008), 3) The Double-Weighted Algorithm proposed by Elkan and Noto, (2008), 4) The biased SVM method proposed by Liu et al. (2003), 5) Our $\text{SNOBProb}_{\text{ElkNot}}$ algorithm, 6) our $\text{SNOBProb}_{\text{Neg}}$ algorithm, and lastly 7) our $\text{SNOBProb}_{\text{raw}}$ algorithm. In all cases, results are presented after performing 10-fold cross validation, with the same training-test splits used for all 7 algorithms on each fold. In addition, each 10-fold cross validation experiment is performed 5 times, with different random cross-validation splits, with the results averaged together. This is done to ensure that no algorithm gains an unexpected advantage via some unknown characteristic of a particular 10-fold cross-validation split.

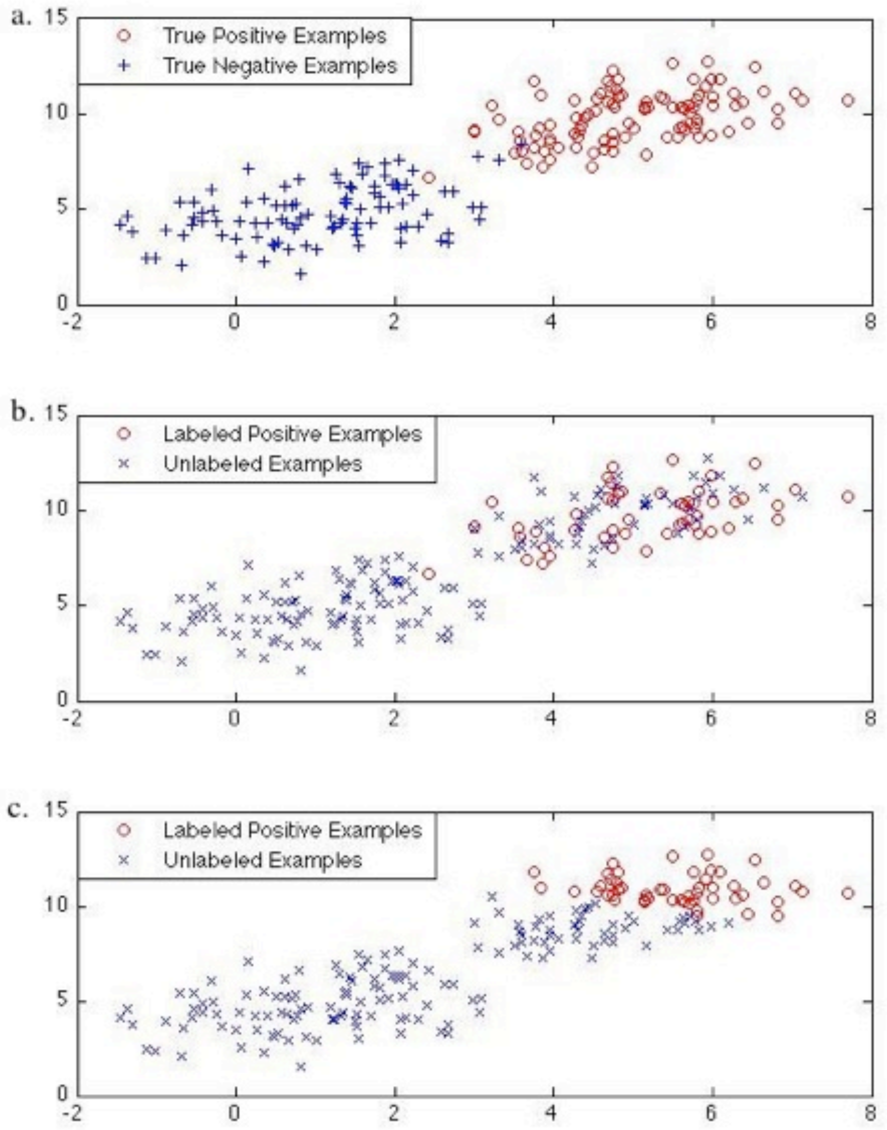


Figure 3.11 Sample synthetic data (a.), generated from two Gaussian distributions. The bias of

the labeled positives can be varied from selected uniformly at random (b.), or according to a bias function that systematically labels only positive examples that are as far away as possible from the decision boundary between true positives and negatives.

3.5.6 Synthetic Data Results

We compare the 7 algorithms described in section 3.6.4 on 3 different scenarios using synthetic data. Each scenario has a different cardinality for *p*, the number of labeled positive examples, *q* the number of unlabeled positive examples, and *n*, the number of unlabeled negative examples.

In the first scenario (results presented in figure 3.12), we set $p = 400$, $q = 100$, and $n = 400$. Thus we have a scenario in which the majority of true positives are already known, but algorithms must seek to discover the remaining positives from amidst the unlabeled negatives. All algorithms performed well in this scenario, but with the performance of the two algorithms laid out in Elkan and Noto 2008 degrading as the percentage of labeled positive examples chosen via the bias function described in section 3.6.4 approached 100%. Our own SNOBPROB-Neg algorithm performs better at low biases, but also suffers from degradation as the bias approaches 100%. The baseline BiasSVM method of Liu et al. (2003) exhibits performance stronger still, yet again suffers degradation at the highest bias thresholds. Lastly, our SNOBPROB-raw method does not perform as well with low bias percentages, but does achieve the highest performance at the highest biases. We do not show the results of the area under the receiver-operator characteristic curve (AUC_ROC) for this scenario, as the differences in values between algorithms were negligible.

In the second scenario, we set with p = 250, q = 250, and n =500, creating a scenario where there are just as many unlabeled positive examples as labeled. As shown in figure 3.13, the results once again show accuracy degradation correlated with the bias percentage, with the method of Elkan and Noto (2008) and Liu et al. (2003) affected more strongly than our proposed methods (other than SNOBPROB-Neg, which also suffers from bias degredation). In particular, our SNOBPROB-Raw algorithm maintains near-optimal accuracy up until 100% bias, where accuracy drops but still remains higher than the other algorithms. In terms of AUC_ROC, most of the algorithms performed near-optimally for the entire bias range.

The third and final scenario sets p = 100, q = 400, and n =500, and is arguably the most "difficult" scenario, as there is the least amount of labeled information. Figure 3.14 depicts the results, showing that once again, our SNOBPROB-Raw algorithm achieves near optimal accuracy results up until 100% bias, where it still performs the best. The AUC_ROC results are less conclusive, with many algorithms again achieving near-optimal results. Specifically, the BiasSVM method shows the strongest performance, followed by our SNOBPROB-Raw and SNOBPROB-Neg algorithms.

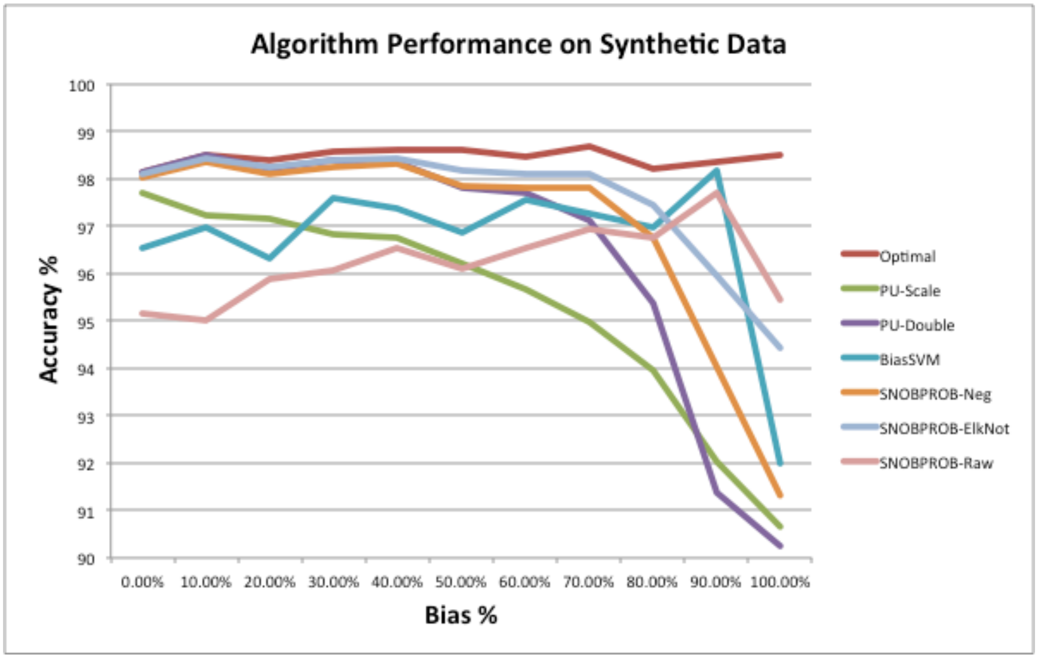
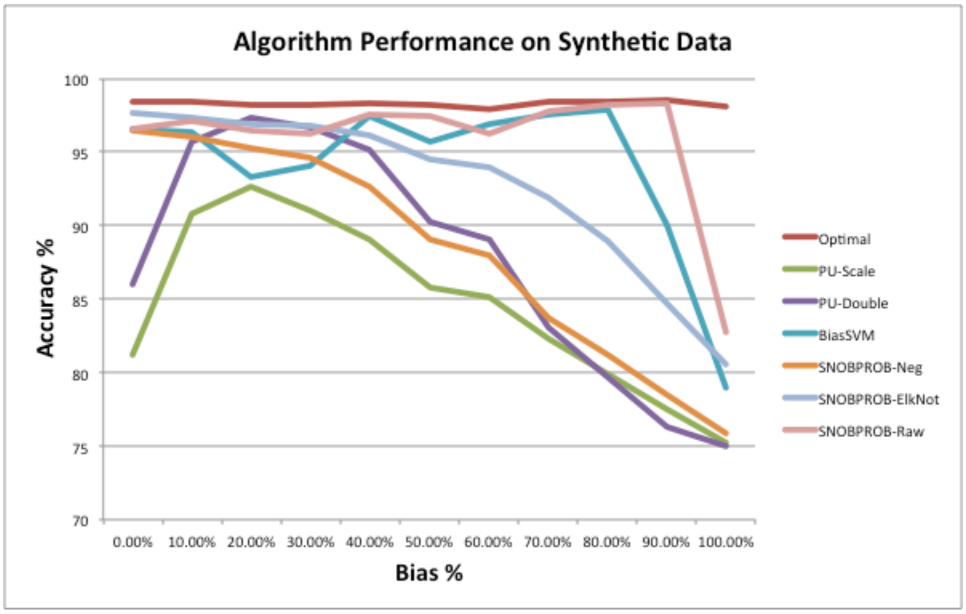


Figure 3.12 Accuracy results on a synthetic dataset with p = 400, q =100, and n =500, with a sliding scale of bias in the method in which the labeled examples were picked from all true positives (see section 3.6.4 for a description of synthetic data generation).



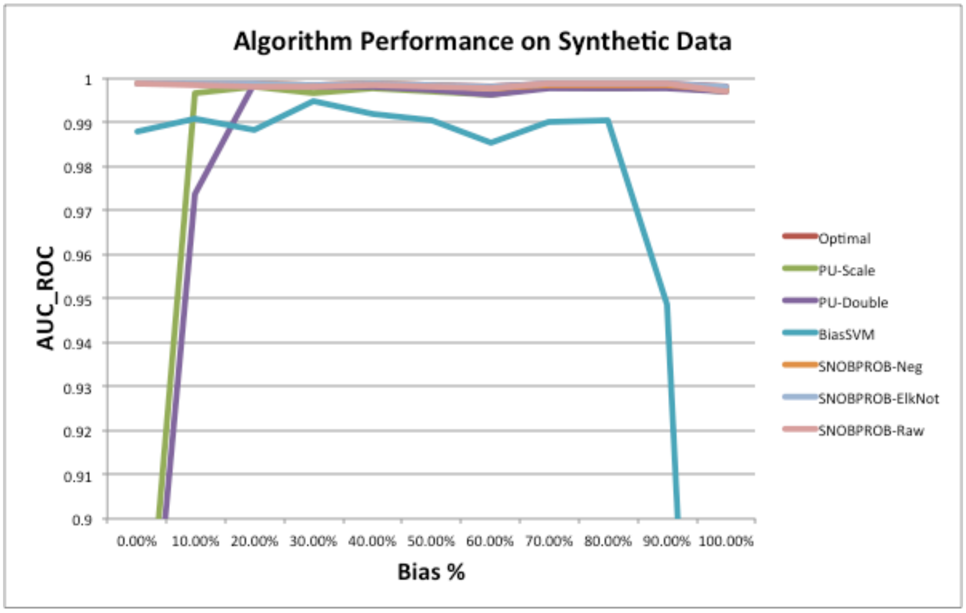
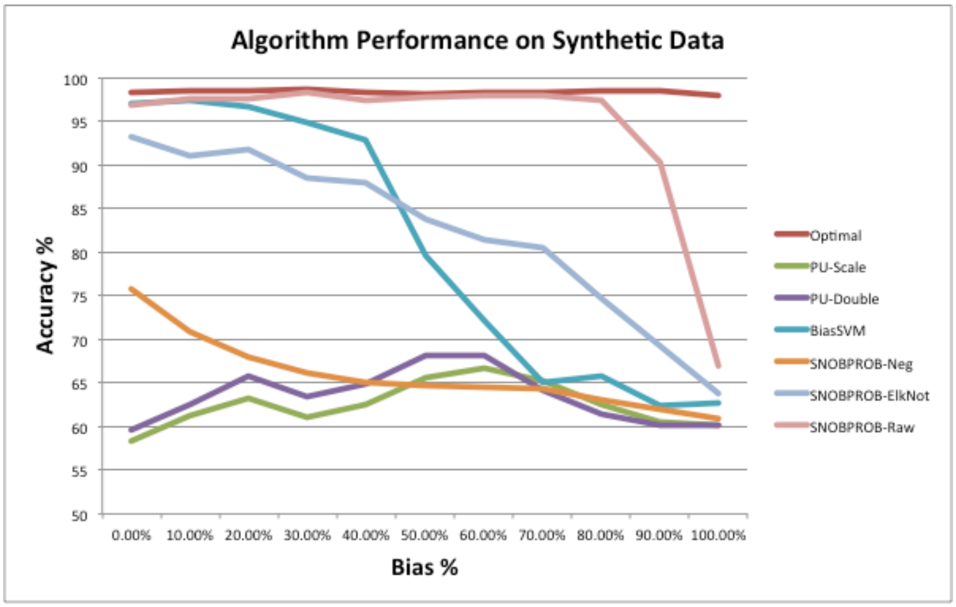


Figure 3.13 Results similar to figure 3.12, but with both accuracy and area under the Receiver-Operator Characteristic curve, for synthetic data with $p = 250$, $q = 250$, and $n = 500$.



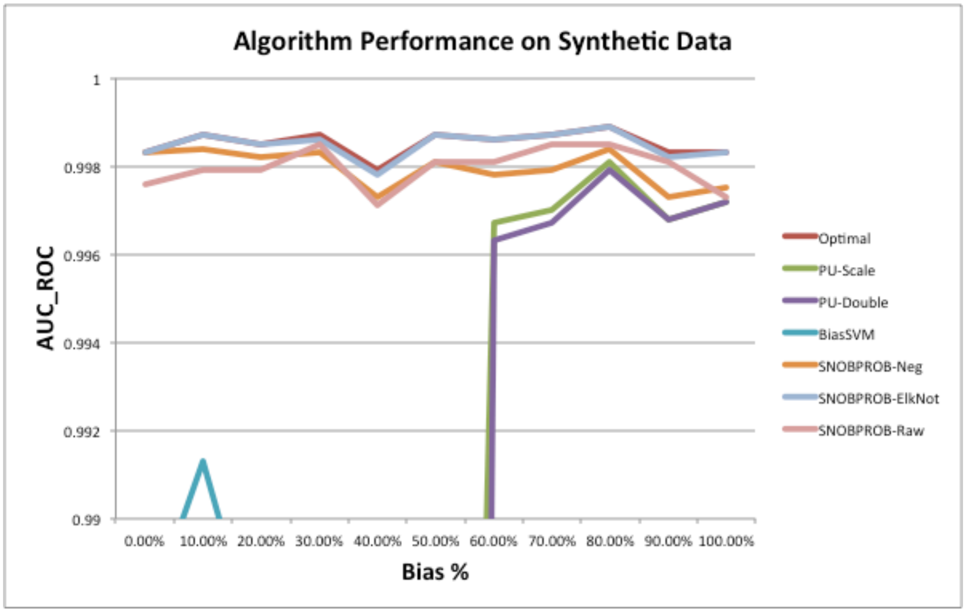


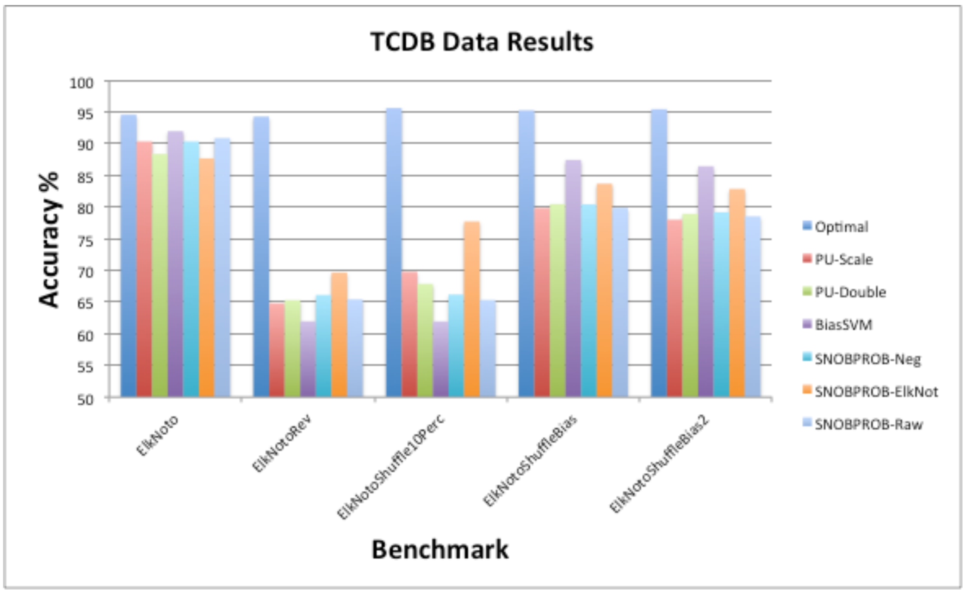
Figure 3.14 Results similar to figure 3.13, but with p = 100, q = 400, and n = 500.

3.5.7 TCDB Data Results

We also present results on real-world data, obtained from Elkan and Noto (2008). This data, comprised of SWISS-PROT records from the TCDB database (Saier et al., 2006), which contains 2453 labeled positive examples, 348 unlabeled positive examples, and 4558 unlabeled negative examples. We test the same 7 algorithms as in 3.6.5, first on the original dataset (which we call ElkNoto), then on the dataset with P and Q reversed, so that now only 348 examples are labeled, and the other 2453 are unlabeled (we refer to this scenario as ElkNotoRev). Additionally we create a scenario where the original P and Q are shuffled together, and then P' is chosen as 10% of the true positive examples, with Q' comprised of the remaining 90% (we refer to this scenario as ElkNotoShuffle10Perc). Finally, we test on two scenarios in which a random positive record is chosen, and the rest of the labeled positive examples are chosen with bias, such that records whose correlation between their *tf-idf* vector representation, and that of the randomly selected record, is highest. In this two biased scenarios, we maintain the relative sizes of P and Q as the original TCDB data.

In terms of accuracy, the strongest algorithms are the BiasedSVM algorithm, as well as our SNOBPROB-ElkNot algorithm, with our algorithm performing best on the ElkNotoRev and ElkNotoShuffle10Perc scenarios (see figure 3.15). This suggests that our algorithm achieves greater accuracy when the cardinality of P is much smaller than that of Q… i.e. there are many more unlabeled positive examples than labeled ones. This corroborates the findings of section 3.6.5. In terms of AUC_ROC, however, the BiasedSVM algorithm seems to perform the best across the board, even in the biased scenarios, which is less congruous with the results of 3.6.5.

Our SNOBPROB-Neg and SNOBPROB-Raw algorithms are competitive with, or perform better than the methods proposed in Elkan and Noto (2008), in terms of the AUC_ROC metric.



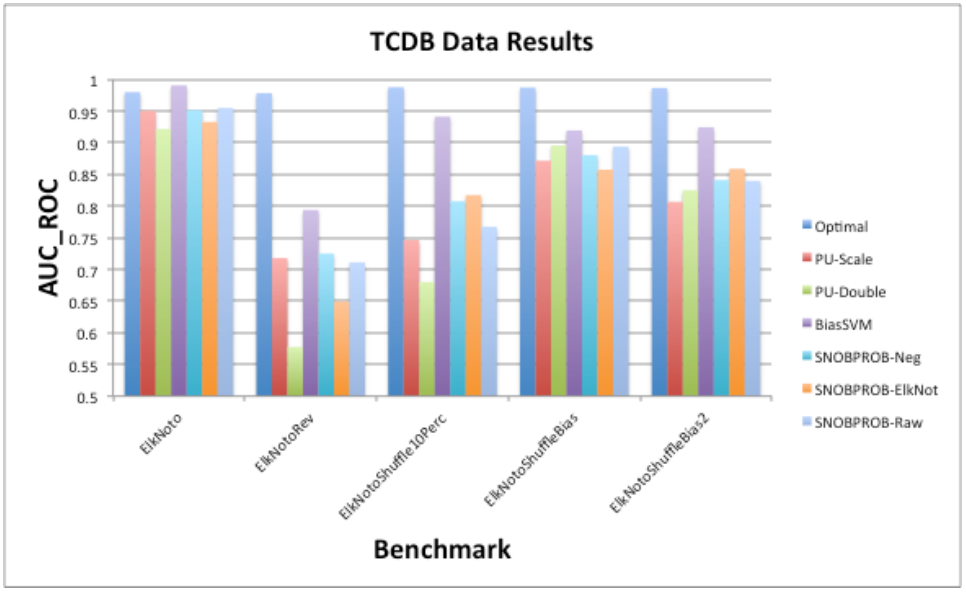


Figure 3.15 Results on different scenarios based on the TCDB database data (see section 3.6.6).

4. Tertiary Structure as a Predictor of Function

4.1 Motivation

It has been well-documented that the shape (also called the fold) of a protein often determines many of the functions that the protein will perform (Bonneau et al., 2004; Malmstrom et al., 2007; Zhang et al., 2009). Yet despite the fact that we have fully sequenced the genomes of many organisms, our knowledge of protein folds is still relatively sparse. Even though the underlying physical properties and interactions of proteins at the atomic level are well-understood, the sheer number of atoms involved, as well as the fact that is impossible to know *a priori* whether pairs of atoms far away from each other in sequence space may end up near each other in physical space, makes the protein-folding problem quite difficult. In fact, the computational costs of the protein-folding problem are just as great, if not greater, than the function prediction problem.

That being said, there does exists a collection of known folds of proteins, many of which have been assigned functions (Berman et al., 2000), as well as tools for computationally estimating the fold of proteins based on their sequence (Rohl et al., 2004). The last necessary component is an efficient algorithm to compute the similarity of two protein structures, for which we use MAMMOTH (Ortiz et al., 2002). With these tools, we can incorporate structural data as a valuable input feature for our function prediction algorithm.

4.2 HPF pipeline

The Human Proteome Folding pipeline is a procedure that matches sequences to known structures in the Protein Data Bank (Berman et al., 2000), or attempts de novo predictions of un-matched sequences via Rosetta (Rohl et al., 2004). The pipeline, developed by Drew et al. (2011) begins by cutting the query sequence into predicted domains, via the Ginzu algorithm (Chivian et al., 2003). The domains are then run through multiple algorithms and heuristics (see Drew et. al 2011) to match domains to known sequences. If no such match can be found, and the domain is of an appropriate size, a de novo structure is generated for that domain utilizing Rosetta on the IBM World Community Grid **(**www.wcgrid.org). The number of de novo structures, known as decoys, that are generated for each domain range between 25,000 and 100,000, depending on the available computing resources on the IBM WCG. The resulting decoys are then clustered, and the top five most populous clusters are stored for use in creating the structural similarity data (see section 4.3).

4.3 Turning structure into functional similarity

We begin by precomputing the pairwise similarity scores of all structures in the Astral database (Chandonia et al., 2004), which is a subset of the Protein Data Bank whose structures have been mapped to specific domains rather than simply whole proteins. Then, to calculate the structural similarity of the human proteins represented by astral structures from the PFP, we simply lookup the previously calculated pairwise similarity scores (MAMMOTH z-scores). The structural similarity of two humans proteins is defined as the sum of the maximum pairwise similarity score between the structures representing each protein averaged over the total number of structures representing both proteins, as described by the following equation:

$$similarity(p_1, p_2) = \frac{\sum_{i \in p_1} \max\{structSim(i,j) \mid j \in p_2\} + \sum_{j \in p_2} \max\{structSim(j,i) \mid i \in p_1\}}{|p_1| + |p_2|}$$

where *P1* and *P2* are the sets of all astral structures representing two proteins, *p1* and *p2*, respectively, and *structSim(i, j)* is a function returning the structural similarity score (MAMMOTH z-score) of two astral structures *i* and *j*.

For proteins that do not match to a known structure in the pdb, but have *de novo* structure predictions, the situation becomes slightly more complex. Each *de novo* prediction is actually a multitude of predictions, known as "decoys", which are then clustered to provide more predictive stability. We utilize the top 5 decoys for each domain with a *de novo* prediction, taking the maximum Z-score for any of the member decoys. Thus if a *de novo* domain is being compared to an astral-matched domain, then the *structSim(i, j)* function will be the maximum of 5 MAMMOTH z-scores, while if both domains are *de novo* structures, then *structSim(i, j)* will be the maximum of 25 MAMMOTH z-scores.

5. Case Studies

5.1 RNA-Binding case study

5.1.1 Function Prediction

Predictions for the Gene Ontology (GO) Molecular Function term RNA binding, along with first-generation child terms of RNA binding, were calculated and compared to a set of experimentally verified but as-yet unannotated RNA binding proteins discovered by Baltz et al., 2012. For this work we combined several network types to make function predictions including: i) a network of GO-process and localization similarities, ii-iv) similarities in InterPro and Pfam domain content, v) protein-protein interactions, vi) co-expression relationships, and vii) structural similarity derived from the Proteome Folding database (Drew et al., 2011). Each node of the graph is a gene which may be previously known to have the function in question, known to not possess that function, or may be unlabeled (here we focus on RNA-binding, its child GO-functions, and DNA-binding). Once labels have been propagated on this composite network, discriminant thresholds are chosen to assign predictions to unlabeled sequences.

5.1.2 Data Sets used for function prediction and network figure generation

Our function prediction algorithm makes use of three categorical data types (InterPro family, Pfam family, and GO Biological Process and Cellular Component annotation), a protein-protein interaction network, a co-expression network, and a structure-similarity network for a total of 6 raw data-types. Only the top 100 similarity scores were kept for each sequence and in each data-type, in order to keep the networks sparse, but in the case of PPI data, the sparsity was much greater as the average number of interactions for a sequence that had any know interactions was only 18.

Categorical Data

For each categorical data-type, we create a binary feature vector whose length is the total number of unique categories that appeared in any of the sequences. As in (Mostafavi et al., 2008), we transform this binary vector by turning all 1's into –log(B), and all 0's into log(1-B), where B is the proportion of sequences that have the given feature, thus allowing rarer features to contribute more in the similarity calculation. The network is then constructed from this transformed feature matrix by taking the pairwise Pearson Correlation Coefficients. InterPro and Pfam results were obtained by querying the 49,518 non-redundant sequences against the InterPro database, Release 34.0, (Hunter et al., 2011). GO annotations were obtained from querying the known GI numbers of the sequences against the AgBase GO Retriever (McCarthy et al., 2006).

Gene Expression Data

Gene expression data was obtained from two assays: HG-U133_Plus_2, and U133AAofAv2, which combined have a total of 368 cell types/conditions. The data for each assay was normalized individually using the Affy library in R, and the resulting two expression vectors for each gene were concatenated into one vector. Since expression data is collected at the gene-level, we had to map our sequences to gene names that appeared in the two assays. The network was then created as the pairwise PCC of expression vectors.

Protein-Protein Interaction Data

Protein-protein interaction data was collected from the BioNetBuilder project (Avila-Campillo et al., 2007). The network was left as a binary network, with a 1 if two proteins interacted and a 0 otherwise.

Structure Similarity Data

To measure the structural similarity of the 49,519 human protein sequences, including novel RNA-binding protein sequences, we chose to represent each protein by a collection of astral structures (Brenner et al., 2000). As a first step, we computed the pairwise structural similarity of all astral structures using the MAMMOTH comparison method described in (Ortiz et al., 2002). After having chosen the astral structures that represented each human protein, we then calculated the structural similarity between two human proteins by combining the similarities of their component representative structures.

To represent each human protein with a set of astral structures, we queried them against the Protein Folding Project (PFP) database, which contains protein sequences previously annotated with structure (Drew et al., 2011). We did this by blasting the set of 49,519 human and RNA-binding proteins against a PFP database comprised of the proteomes of six organisms: human, mouse, *Caenorhabditis elegans*, *Escherichia coli*, *Saccharomyces cerevisiae*, and *Arabidopsis thaliana*. For each human protein sequence, the best 250 blast results were retained and then further filtered to keep only those matching sequences with at least 50% identity over 80% sequence length. We then considered the structural annotations of each protein sequence in the filtered blast results, and chose to represent the query human sequence with the astral structures from the best blast match with the most complete structural annotation. In this way, each of the 49,519 human proteins (including novel RNA-binding proteins) was represented by a collection of astral structures.

Roughly 23,000 human proteins remained after this conservative filtering while the remaining 26,000 did not match confidently to structurally annotated sequences in the PFP. Structures for matching proteins were utilized to calculate pairwise similarity scores (as described in section 4.3). If the structural similarity score of a source and target human protein was in the best 100 scores for that source protein, the score for the pair was added to the structure all-vs.-all matrix. In this way we calculated the all-v-all structural similarity matrix representing the set of human and RNA-binding proteins.

5.1.3 Association Network Combination Nuances:

In this particular case study, each node in the network represents a sequence, and as some data-types contain information at the sequence level, and others at the gene level, the coverage of each data-type is not consistent. Additionally some data-types are simply more comprehensive, such as InterPro, which returned results for 38,396 sequences, compared to Pfam, which only returned hits for 35,082. Because the objective function rewards low similarity for negative example pairs, a data-type with less coverage and therefore more sparsity can produce an unfair weight boost in the final network as it avoids edges between positive-negative pairs in the Omega matrix. To remedy this problem, in this case study we only construct Omega from pairs of omni-reachable sequences, where a sequence is defined as omni-reachable if it is in a row that contains at least one non-zero entry from each data-type. If a data-type is dropped by the algorithm due to a negative weight assignment, the set of omni-reachable sequences is re-calculated given the remaining subset of data types (and can only grow larger by doing so).

5.1.4 Predictive Power of Each Data Type

Without the normalization step (described above) the relative magnitude of the weights assigned during the network combination phase of the algorithm could be used as a proxy for relative contribution of each data type to the prediction for each function label. However, the sparsity of different data types plays a major role in the normalization, as the magnitude of the row and column sums will be smaller for sparser data. This complicates the interpretation of the weights as a measure of particular data's predictive power since the average magnitude of edges in each data type differs even before the weights are applied. Therefore in order to access the contributions of each data type in a meaningful manner, we multiply the weights produced by the algorithm (at the network combination stage) by the average non-zero affinity for each data-type, shown below:

Interpro Domains	Pfam Domains	GO Process and Localization	Gene Expression	PPI	Structure
0.0057	0.0055	0.0050	0.0054	0.0427	0.0056

Table 5.1 Relative contribution of each component data type to the final affinity network.

This product yields the average non-zero contribution of each data type to the final affinity matrix used in the label-propagation step (which is itself also normalized). These relative contributions can be seen in table 5.1 for all of the GO function terms predicted.

5.1.5 Discriminant Thresholds through Cross Validation

Once the discriminant vector is calculated, a threshold is required in to make predictions. Threshold values for the discriminant were obtained through k-fold cross-validation. For each of the k calculations, the known labels are dropped on a random leave-out set of size 49,518/k, which contains the same proportion of positive, negative, and unlabeled sequences as the entire set. The discriminant threshold is then varied until the desired precision level is met on the leave-out set, and the recall value for the discriminant threshold is noted. If the desired precision level is unattainable for any discriminant threshold value, then that particular cross-validation run is not counted in the final totals.

Once cross validation is complete, the discriminant threshold value for a given precision is calculated as the average of values for all of the cross-validation tests. We chose to predict functions at precision levels of 80%, 50% and 20%, and set k=10 for the functions of RNA binding and DNA binding, but k=5 for the children of RNA binding to allow for enough positive labels in each of the leave-out sets. Table S2A shows the recall values at each precision for the different function labels.

5.2 CAFA Challenge

5.2.1 CAFA Challenge Description

The Critical Assessment of Functional Annotation (CAFA) challenge is a bi-annual curated competition, whose goal is to evaluate the computational methods of participating groups on several benchmarks and scenarios. All evaluation is done via temporal holdouts, which is recognized as the best available method to combat evaluation issues arising from the PU learning scenario (see section 2.4.3). In the first CAFA challenge, in 2012, 54 methods were evaluated, many showing strong performance on a benchmark of 866 proteins in 11 different organisms, although certainly with room left for improvement (Radivojac, et al., 2013).

In the 2nd CAFA challenge, target sequences for prediction were provided in 27 organisms, spanning the three domains of biological classification, totaling more than 100,000 sequences. The entry submission deadline was closed in January, 2014, with evaluations beginning in July, 2014, utilizing the annotations that had been added to any of the target sequences in the time period after submissions were closed.

Predictive methods were scored according to two metrics. The first is the F-measure, which is a standard metric for machine learning algorithms (Powers, 2011). The harmonic mean of precision and recall, the F-measure captures some of the information contained in each, with the final score being the maximum F-measure achieved along the precision-recall curve. The second measure comes from Information Theory, and is known as the minimum semantic distance (Rada et al., 1989). This measure takes into account the hierarchical structure of the Gene Ontology (GO) tree, and thus the fact that some functional predictions are more specific than others.

Both metrics were applied in two modes: mode 1 only evaluates sequence targets which are completely unannotated in every branch of GO, while mode 2 allows sequences to have existing annotations, so long as they are in a branch other than the branch which is being evaluated. Sequences that had any annotation in the branch being evaluated, even if that annotation was extremely general, were not considered in any of the assessments. In addition, benchmarks were broken down by the source of the annotations into three groups: those coming from the SWISS-PROT database (Bairoch and Apweiler, 1997) , those from the EBI database (Guenter et al., 1997), and those coming from all databases (SWISS-PROT, EBI, UNIPROT-GOA (Dimmer et al., 2012), and GO (Ashburner et al., 2000)).

5.2.2 CAFA Challenge Results

At the time of this publication, only preliminary results were available from the CAFA 2 challenge. These results were based on annotations added to the Gene Ontology between the submission close and the evaluation date, with the number of newly annotated proteins in each benchmark and mode (see section 5.2.1 for a description of evaluation modes) presented in table 5.2. In addition, evaluations were performed for all organisms together, and for just the subset of Eukaryotes as well.

Benchmark	Branch	Mode 1	Mode 2
SWISS-PROT	Molecular Function	232	285
	Biological Process	410	582
	Cellular Component	608	393
EBI	Molecular Function	238	239
	Biological Process	281	157
	Cellular Component	560	306
SWISS-PROT + EBI + UNIPROT-GOA + GO	Molecular Function	656	667
	Biological Process	773	751
	Cellular Component	991	637

Table 5.2

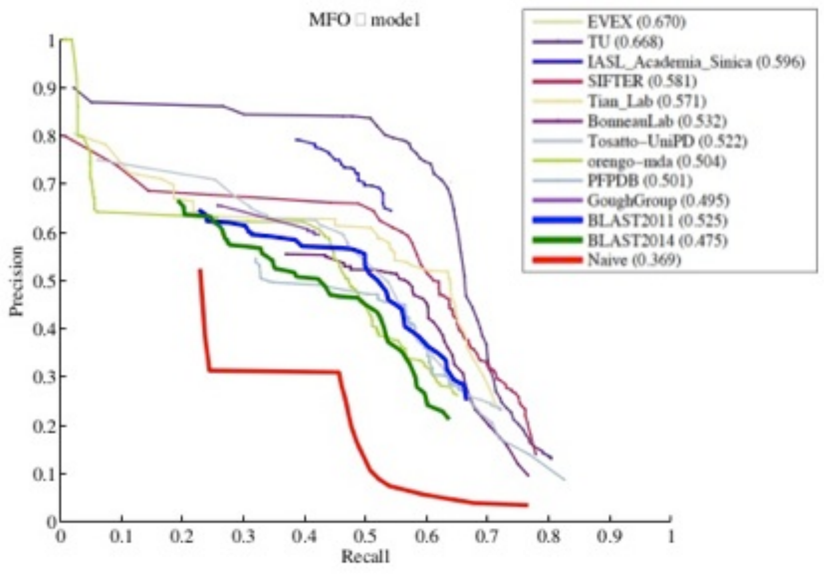
Number of sequences used in each benchmark, for each mode and branch of GO.

Out of the roughly 50 competing algorithms (not every team submitted an entry that could be evaluated on all benchmarks), we placed in the top 10 scores on several, but not all benchmarks and modes. Specifically, on the EBI benchmark, we placed 6th in Molecular Function, 2nd in Biological Process, and 8th in Cellular Component, while on the SWISS-PROT benchmark, we placed 9th in Cellular Component (see figure 5.1).

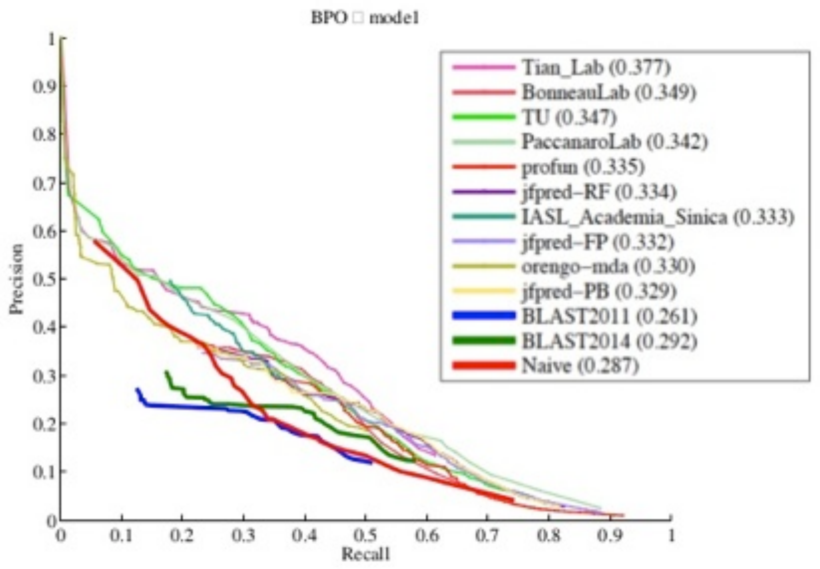
In figure 5.2, we see a comparison of the two modes of evaluation (see section 5.2.1) on the Biological Process EBI benchmark. Our submission performed better in mode 2, which is not surprising, as the presence of existing annotations in other branches of GO helps to inform the prior biases we use for label propagation (see section 2.3.1). This also suggests that our method would do well in a scenario in which annotations in the branch being predicted were also permitted, but more specific predictions were sought.

While further evaluation results are pending from the CAFA assessors, it appears that our methods are highly competitive with the current state of the art protein function prediction algorithms employed by many research teams. In addition, we believe our methods would perform even more strongly in a scenario not considered in CAFA: that of predicting more specific or complete annotations for partially annotated proteins.

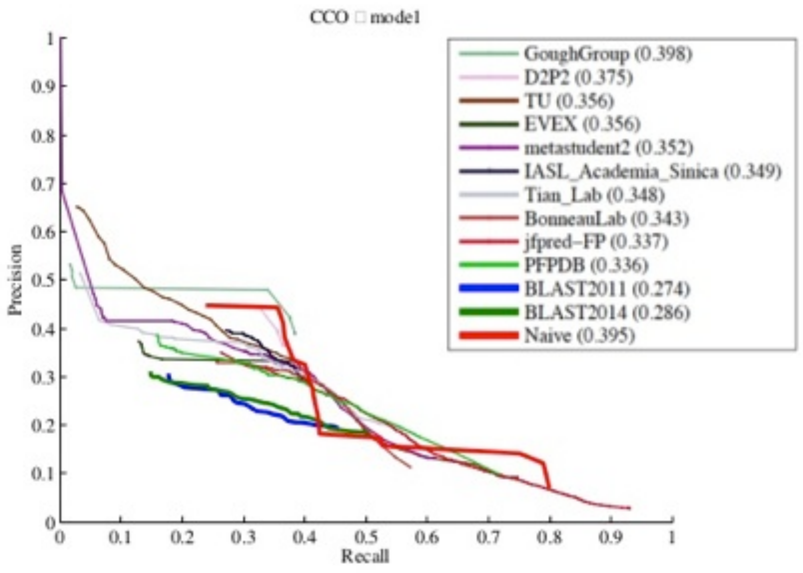
EBI: TOP 10 MODEL 1S



EBI: TOP 10 MODEL 1S



EBI: TOP 10 MODEL 1S



SWISS-PROT: TOP 10 MODEL 1S

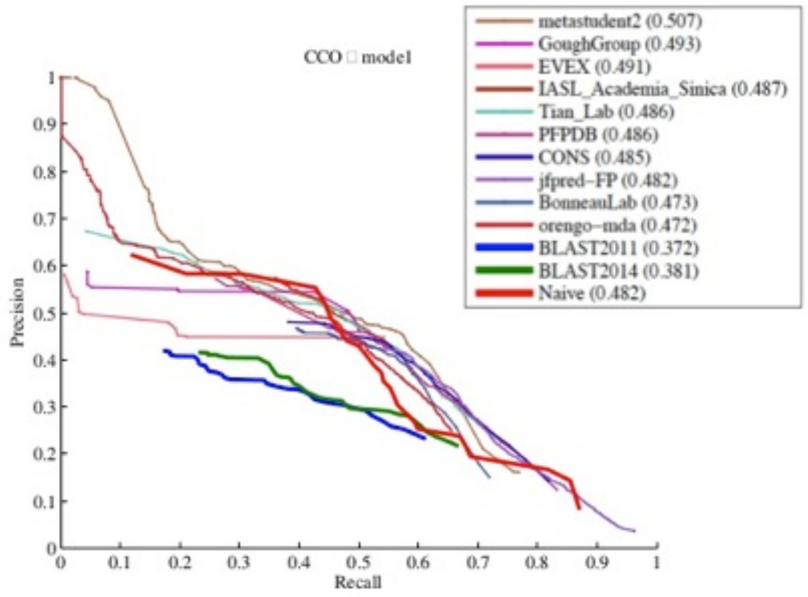


Figure 5.1 Precision-Recall curves for the top 10 methods on several different CAFA2 benchmarks. Methods are ranked by F-measure (shown in the legend), and also compared to 3 baseline methods: Blast2011, Blast 2014, and a Naïve prediction method based on choosing nodes with the most existing annotations.

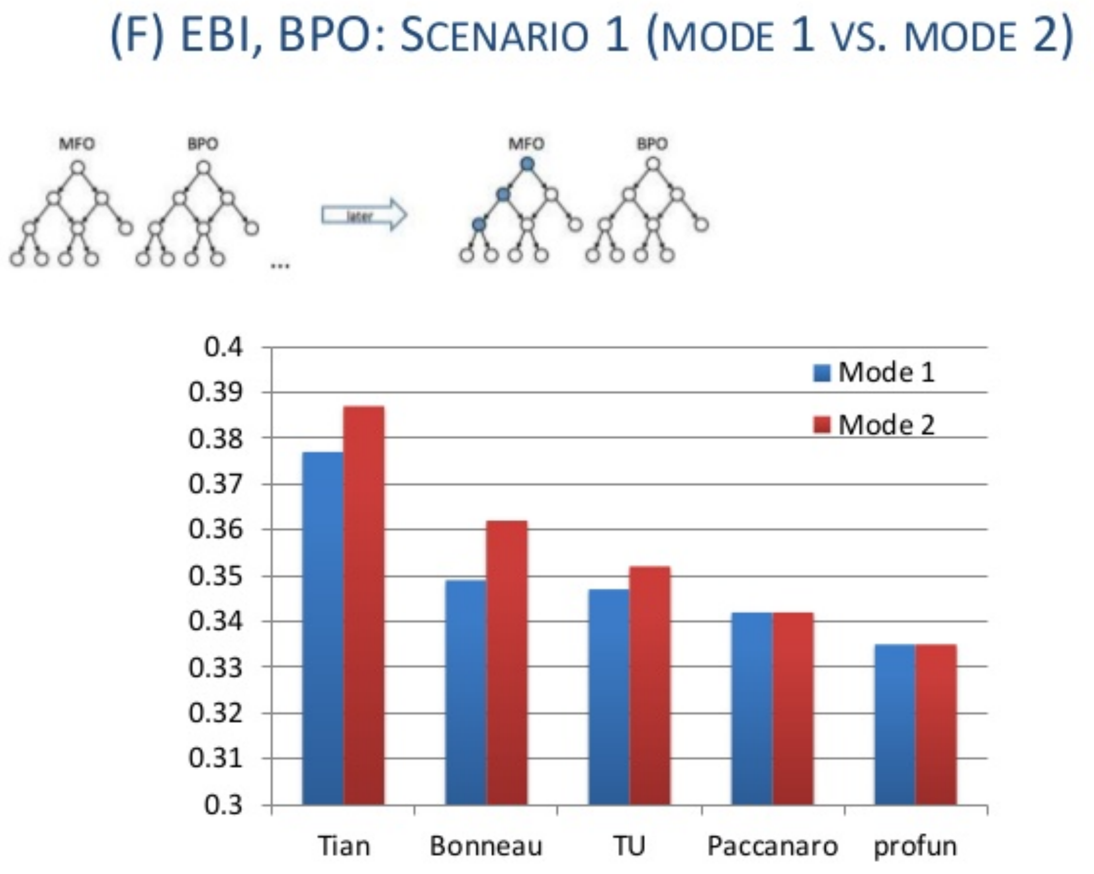


Figure 5.2 Differences in F-measure for the top 5 performing methods in Biological Process on

the EBI benchmark of CAFA2. The F-measures of mode 1 and mode 2 are compared (see section 5.2.1 for a description of the modes).

6. Future Work

While the work here represents several steps forward in the application of positive-unlabeled learning to the protein function problem, there is still much more to be done.

6.1 Protein Function Prediction

The advancements in protein function prediction scope and accuracy presented in this work thus far open the door for additional avenues of exploration. Some of these avenues pertain to additional data types that show promise for inclusion in the feature set, while others focus more on the framework of the learning problem. As the percentage of annotated proteins across all sequenced genomes is still extremely low, further research into function-prediction improvement has the potential to greatly advance the field.

6.1.1 Prediction reconciliation

In the current paradigm, the protein function prediction problem is treated as a series of binary classification problems, learning one function, for one species, at a time. While our work has already begun to address the potential for multi-species prediction (see section 2.7), we have not yet touched upon one of the obvious problems with single-function learning: namely that the binary classification problems are not independent. Since functional terms are described by the

Gene Ontology (GO), which is organized as a tree, predictions for certain GO terms automatically imply predictions for other GO terms at the same confidence level (namely, all ancestral terms).

Previous work has looked into several possibilities regarding multi-function learning. Mostafavi and Morris (2009), experiments both with a framework for learning all functions within a branch of GO simultaneously, as well as reconciling predictions after the fact with Isotonic Regression, and also predicting sequentially from most general to most specific functional terms, or vice-versa, and using the results of predictions for ancestral or child terms as priors for the next round of predictions. Unfortunately, Mostafavi and Morris (2009) found little benefit in any of these approaches, and most machine learning methods for function prediction still proceed under the binary paradigm.

We believe that one potentially fruitful avenue of research for this problem, is the utilization of a Bayesian Network to reconcile predictions. This approach has been attempted before for the PFP problem itself (Leone and Pagnani, 2005; King et al., 2003), but to our knowledge has never been applied to the reconciliation of predictions made by another method. One potential embodiment would proceed gene by gene, with each node in the Bayes Net representing a GO term that that gene received a non-negligible discriminant score for. Edges in the network would mirror the edges in the GO tree, with the conditional probabilities dependent upon the discriminant values for each GO term calculated for that gene during the course of binary function prediction. The maximum a posteriori probability estimate for this network would then represent the most probable assignment of function terms, according to the original binary discriminant values, but which still obey the hierarchy of the Gene Ontology.

6.1.2 More function-specific data combination

A key component of the function prediction methodology described in this work, is the network combination step, by which heterogeneous networks obtained from different data types are combined into one functional similarity network (See section 2.3.3). In Mostafavi and Morris (2010), as well as Youngs *et al*. (2013), various methods were explored to reduce the potential for over-fitting this combination to one specific function, while still maintaining some type of specificity. In both works, experiments indicated that training the network combination algorithm on all functions in a given branch of GO simultaneously provided the best results.

It is quite intuitive, however, to imagine that different types of functions would be better observed through different types of data. While Gene Expression may be more revealing about the presence of function *a*, for example, function *b* might be more dependent on protein-protein interaction, and therefore PPI data should play a greater role in the final association network. We believe greater exploration into learning the association network based upon subsets of functions is warranted, making use of the GO structure to train on enough simultaneous functions to avoid over-fitting while still preserving some of the data-type specificity inherent in different functions.

An additional aspect of the function-specific data combination is the tertiary structure data (see section 4.3). While the tertiary structure of a protein represents a huge amount of information, it is distilled down into a single number per pair of proteins. Even in the simplest case: a single-domain protein, this similarity number might be skewed by discrepancies in a part of the protein that is not functionally relevant. In multi-domain proteins, it is certainly the case that structure-similarity information is being diluted, as many functions can be associated with just a single domain. One potential approach to preserve more of the information inherent in tertiary structure is to predict functions at the domain, rather than the protein level (see section 6.1.3). Another, however, is to apply a similar algorithm to the network combination algorithm, specifically to the generation of the structural-similarity association network. By utilizing known annotations for a function, or more likely a subset of ancestral functions in a branch, different structural-similarity feature matrices would be created that utilized only the structural similarity information from domains identified as associated with those functions.

6.1.3 Domain-centric efforts

As mentioned in section 6.1.2, many functions can be characterized at the domain level, rather than the protein level. It would therefore be logical to choose the granularity of the function prediction network such that each node was a domain, rather than a protein. Unfortunately, not all data is available at the domain granularity (Protein-Protein Interaction data, for example). Additionally, the training labels (existing GO annotations) also exist only at the protein/gene granularity, and not at the domain specificity. The second issue has begun to be addressed in the literature (Fang and Gough, 2013), but the first presents some interesting opportunities for our function prediction framework.

When predicting at the domain specificity, it becomes necessary to distribute similarity edges from protein-granular data types across all of the domain nodes within that protein. The weight of the data cannot simply be assigned to each node (as then proteins with more domains would exert undue influence), but must rather be divided amongst the domain nodes. This allows for the potential to incorporate the uncertainty inherent in methods that attempt to assign labels or data to specific domains, by allowing the distribution of the weight to be reflective of the confidence of the domain assignment within that protein.

6.1.4 Phylogenetic Tree incorporation

Many computational function-prediction methods utilize phylogeny as an input (Eisen, 1998; Englehardt et al., 2005; Gaudet et al., 2011). As these methods demonstrate, the evolutionary process captured in a phylogenetic tree can be quite informative about protein function. While we do include phylogeny as an input feature for our function prediction methodology, we do so only in a limited fashion: namely using phylogenetic profiles. These are simply binary vectors indicating whether or not the protein in question has a homolog in each of a set of reference species, which is far less informative than a full phylogenetic tree.

In order to create a phylogenetic-tree-based input data type, an accurate computational phylogenetic tree generation tool is needed, whose complexity is not prohibitive (Engelhardt et al., 2005). Once trees are generated, protein-protein similarity scores must be generated. The similarity of trees is a concept already explored in the literature, mostly in the context of text-classification. Some potential algorithms for transforming the phylogenetic trees into an input association matrix include those presented in Culik and Wood (1982), and Xue et al., (2008).

6.2 PU-Learning in Biased Scenarios

As discussed in section 3.6, a common situation in PU learning is one in which the set of labeled positive examples are chosen from the complete set of positive examples according to some bias function, rather than uniformly at random. We have presented novel algorithms that perform well in this scenario, but an interesting avenue of future exploration is one in which the nature of the bias itself is incorporated. In certain examples, such as protein function prediction, the nature of the labeling bias is understood. If that bias could be estimated with accuracy, then equation 3.1 provides a methodology for learning in this scenario.

Lastly, the performance results presented in sections 3.6.5 and 3.6.6 indicate that different algorithms might prove more or less appropriate depending upon characteristics of the learning scenario. Namely, on the fraction of true positives that are labeled, as opposed to unlabeled. Elkan and Noto (2008) provide a methodology to estimate this number in the case that labels are chosen uniformly at random, but in the case that labels are chosen with bias, no such method exists. Even so, it may often be that case that domain experts can provide an educated guess in different scenarios, as to the percentage of true positives that remain to be labeled. Thus we believe future work into the nuances of why different algorithms exhibit differing performance on these scenarios is warranted.

7 References

Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, et al. (2000) Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. Nat Genet **25**: 25-29.

Avila-Campillo, I., Drew, K., Lin, J., Reiss, D.J., and Bonneau, R. (2007). BioNetBuilder: automatic integration of biological networks. Bioinformatics **23**, 392-393.

Bairoch, Amos, and Rolf Apweiler (1997). "The SWISS-PROT protein sequence data bank and its supplement TrEMBL." *Nucleic acids research* 25.1: 31-36.

Baltz AG, Munschauer M, Schwanhausser B, Vasile A, Murakawa Y, et al. (2012) The mRNA-bound proteome and its global occupancy profile on protein-coding transcripts. Mol Cell **46**: 674-690.

Berman, Helen M., et al. (2000) "The protein data bank." *Nucleic acids research*28.1: 235-242.

Bhardwaj N, Gerstein M, Lu H (2010) Genome-wide sequence-based prediction of peripheral proteins using a novel semi-supervised learning technique. BMC Bioinformatics **11** Suppl 1: S6.

Blei DM, Ng AY, Jordan MI (2003) Latent Dirichlet allocation. Journal of Machine Learning Research **3**: 993-1022.

Bonneau R, Baliga N, Deutsch E, Shannon P, Hood L. (2004**).** Comprehensive de novo structure prediction in a systems-biology context for the archaea Halobacterium sp NRC-1. Genome Biol 5: R52. doi: 10.1186/gb- 2004-5-8-r52.

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

Cesa-Bianchi N, Valentini G (2010) Hierarchical cost-sensitive algorithms for genome-wide gene function prediction.

Chandonia JM, Hon G, Walker NS, Lo Conte L, Koehl P, Levitt M, Brenner SE (2004). The ASTRAL Compendium in 2004. Nucleic Acids Res 32: D189–D192.

Chivian, Dylan, et al. (2003) "Automated prediction of CASP-5 structures using the Robetta Server." *Proteins: Structure, Function, and Bioinformatics* 53.S6: 524-533.

Dimmer, Emily C., et al. (2012). "The UniProt-GO annotation database in 2011."*Nucleic acids research* 40.D1 : D565-D570.

Drew, K., et al. (2011) The Proteome Folding Project: proteome-scale prediction of structure and function. Genome Res., **21**: 1981-1994.

Eisen, Jonathan A., (1998)."Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis." *Genome research* 8.3: 163-167.

Elkan C, Noto K (2008) Learning classifiers from only positive and unlabeled data. Proc. ACM SIGKDD, pp. 213-220.

Engelhardt, Barbara E., et al. (2005) "Protein molecular function prediction by Bayesian phylogenomics." *PLoS computational biology* 1.5: e45.

Fang, Hai, and Julian Gough (2013) "dcGO: database of domain-centric ontologies on functions, phenotypes, diseases and more." *Nucleic acids research*41.D1: D536-D544.

Gaudet, Pascale, et al., (2011) "Phylogenetic-based propagation of functional annotations within the Gene Ontology consortium." *Briefings in bioinformatics* 12.5: 449-462.

Gomez SM, Noble WS, Rzhetsky A (2003) Learning to predict protein-protein interactions from protein sequences. Bioinformatics **19**: 1875-1881.

Greene CS, Troyanskaya OG (2012) Accurate evaluation and analysis of functional genomics data and methods. Ann N Y Acad Sci **1260**: 95-100.

Guan Y, Myers CL, Hess DC, Barutcuoglu Z, Caudy AA, et al. (2008) Predicting gene function in a hierarchical context with an ensemble of classifiers. Genome Biol **9** Suppl 1: S3.

Stoesser, Guenter, et al. (1999). "The EMBL nucleotide sequence database." *Nucleic Acids Research* 27.1: 18-24.

Hunter, S., Jones, P., Mitchell, A., Apweiler, R., Attwood, T.K., Bateman, A., Bernard, T., Binns, D., Bork, P., Burge, S., *et al.* (2011). InterPro in 2011: new developments in the family and domain prediction database. Nucleic Acids Res **40**, D306-D312.

Huttenhower C., Hibbs M. A., Myers C. L., Caudy A. A., Hess D. C., et al. (2009) The impact of incomplete knowledge on evaluation: an experimental benchmark for protein function prediction. Bioinformatics 25: 2404–2410.

Kim, W. K et al. (2008) Inferring mouse gene functions from genomic-scale data using a combined functional network/classification strategy. Genome Biol **9** (Suppl. 1), S5.

King OD, Foulger RE, Dwight SS, White JV, Roth FP (2003) Predicting gene function from patterns of annotation. Genome Res **13**: 896-904.

Lee, H., et al. (2006) Diffusion Kernel-Based Logistic Regression Models for Protein Function Prediction. OMICS, **13**, 896-904.

Leone, M., and Pagnani, A. (2005) Predicting protein functions with message passing algorithms. Bioinformatics, **21**(2), 239-247. doi:10.1093/bioinformatics/bth491.

Liu B, Dai Y, Li X, Lee WS, Yu PS. (2003) Building text classifiers using positive and unlabeled examples; IEEE. pp. 179-186.

Malmstrom L, Riffle M, Strauss CEM, Chivian D, Davis TN, Bonneau R, Baker D. (2007). Superfamily assignments for the yeast proteome through integration of structure prediction with the Gene Ontology. PLoS Biol 5: e76. doi: 10.1371/journal.pbio.0050076.

Marcotte, E. M., **et al.** (1999) A combined algorithm for genome-wide prediction of protein function. Nature, **402**, 83-86.

McCarthy, F.M., Wang, N., Magee, G.B., Nanduri, B., Lawrence, M.L., Camon, E.B., Barrell, D.G., Hill, D.P., Dolan, M.E., Williams, W.P., ***et al.*** (2006). AgBase: a functional genomics resource for agriculture. BMC Genomics **7***,* 229.

Mostafavi, S., et al. (2008) GeneMANIA: A real-time multiple association network integration algorithm for predicting gene function. Genome Biol., **9** (Suppl. 1), S4.

Mostafavi S, Morris Q. (2009) Using the Gene Ontology hierarchy when predicting gene function; AUAI Press. pp. 419-427.

Mostafavi, S., and Morris, Q. (2010) Fast integration of heterogeneous data sources for predicting gene function with limited annotation. Bioinformatics, **26**, 1759-1765.

Obozinski, G., **et al.** (2008) Consistent probabilistic outputs for protein function prediction. Genome Biol., **9** (Suppl 1.), S6.

Ortiz, A.R., Strauss, C.E., and Olmea, O. (2002). MAMMOTH (matching molecular models obtained from theory): an automated method for model comparison. Protein Sci **11**, 2606-2621.

Östlund, Gabriel, et al. (2010) "InParanoid 7: new algorithms and tools for eukaryotic orthology analysis." *Nucleic acids research* 38.suppl 1: D196-D203.

Pandey G, Myers CL, Kumar V (2009) Incorporating functional inter-relationships into protein function prediction algorithms. BMC Bioinformatics **10**: 142.

Pavlidis, P., and Gillis, J., (2012) Progress and challenges in the computational prediction of gene function using networks. F1000 Research **1**(14).

Peña-Castillo, L., et al. (2008) A critical assessment of Mus musculus gene function prediction using integrated genomic evidence. Genome Biol., **9** (Suppl 1.), S2.

Philip Jones, David Binns, Hsin-Yu Chang, et al. (2014). InterProScan 5: genome-scale protein function classification**.** *Bioinformatics, Jan 2014; doi:10.1093/bioinformatics/btu031.*

Powers, David Martin (2011). "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation."

Puelma T, Gutierrez RA, Soto A (2012) Discriminative local subspaces in gene expression data for effective gene function prediction. Bioinformatics **28**: 2256-2264.

Qi, Y., et al. (2008) Random forest similarity for protein-protein interaction prediction from multiple sources. Pac. Symp. Biocomputing, 531-542.

Rada, Roy, et al. (1989) "Development and application of a metric on semantic nets." *Systems, Man and Cybernetics, IEEE Transactions on* 19.1: 17-30.

Radivojac, Predrag, et al. (2013) "A large-scale evaluation of computational protein function prediction." *Nature methods* 10.3: 221-227.

Rohl, Carol A., et al. (2004) "Protein structure prediction using Rosetta." *Methods in enzymology* 383: 66-93.

Rocchio JJ (1971) Relevance feedback in information retrieval.

M. H. Saier, C. V. Tran, and R. D. Barabote (2006). "TCDB: the transporter classification database for membrane transport protein analyses and information". Nucleic Acids Research, 34:D181–D186.

Smialowski P, Pagel P, Wong P, Brauner B, Dunger I, et al. (2010) The Negatome database: a reference set of non-interacting protein pairs. Nucleic Acids Research **38**: D540-D544.

Smoot, M., et al. (2011) Cytoscape 2.8: new features for data integration and network visualization. Bioinformatics, **27**(3), 431-432.

Stark, C., Breitkreutz, B. J. (2006) BioGRID: a general repository for interaction datasets. Nucl. Acids Res., **34** (Suppl. 1), D535-D539.

Suarjana, M., and Law, K. H. (1994) Successive conjugate gradient methods for structural analysis with multiple load cases. Int. Journal for Num. Methods in Eng., **37**(24), 4185-4203.

Tasan, M., et al. (2008) An en masse phenotype and function prediction system for Mus musculus. Genome Biol., **9**(Suppl. 1), S8.

Troyanskaya, O.G., et al. (2003) A Bayesian framework for combining heterogeneous data sources for gene function prediction (in Saccharomyces cerevisiae). Proc. Natl. Acad. Sci., **100**, 8348-8353.

Tsuda, K., et al. (2005) Fast protein classification with multiple networks. Bioinformatics, **21**(Suppl. 2), ii59-ii65.

Warde-Farley D, Donaldson SL, Comes O, Zuberi K, Badrawi R, et al. (2010) The GeneMANIA prediction server: biological network integration for gene prioritization and predicting gene function. Nucleic Acids Res **38**: W214-220.

Youngs N, Penfold-Brown D, Drew K, Shasha D, Bonneau R (2013) Parametric Bayesian priors and better choice of negative examples improve protein function prediction. Bioinformatics **29**: 1190-1198.

Yousef M, Jung S, Showe LC, Showe MK (2008) Learning from positive examples when the negative class is undetermined--microRNA gene identification. Algorithms Mol Biol **3**: 2.

Yu H, Han J, Chang KC-C. PEBL: positive example based learning for Web page classification using SVM; 2002. ACM. pp. 239-248.

Zhang, C., et al. (2008) An integrated probabilistic approach for gene function prediction using multiple sources of high-throughput data. Int. Journal of Comp. Biology and Drug Design, **1**(3), 254-274.

Zhang Y, Thiele I, Weekes D, Li Z, Jaroszewski L, Ginalski K, Deacon AM, Wooley J, Lesley SA, Wilson IA, et al. (2009). Three-dimensional structural view of the central metabolic network of Thermotoga maritima. Science 325: 1544–1549.

Zhao XM, Wang Y, Chen L, Aihara K (2008) Gene function prediction using labeled and unlabeled data. BMC Bioinformatics **9**: 57.

Zhou, D., et al. (2004) Learning with local and global consistency. Adv. Neural Inf. Process. Syst., **16**, 321-328.

Zhu, X., et al . (2003) Semi-supervised learning using Gaussian fields and harmonic functions. In Proceedings of the Twentieth International Conference on Machine Learning, Washington DC, USA.