**Parametric Bayesian Priors and Better Treatment of Negative Examples Improve Protein Function Prediction**

Noah Youngs, Duncan Penfold-Brown, Kevin Drew, Dennis Shasha, Rich Bonneau

**Abstract**

**Motivation:** Computational biologists have demonstrated the utility of machine learning methods for prediction protein function, in particular the GeneMania algorithm, which combines functional association networks to predict function**(mention GeneMania by name in the abstract?)**. Yet even the best performing function prediction algorithms often rely on heuristics for important components of the algorithm, such as choosing negative examples or values of parameters. The improper choice [treatment] of negative examples, in particular, can hamper the predictive power of such algorithms.

**Results:** We present a novel approach for treating negative examples, which relies on a Bayesian prior computed from observed data. This prior is also utilized in the course of function prediction, and is dependent on several parameters. Accordingly, we present a framework for tuning these and other parameters required by the algorithm. We incorporate a different optimization algorithm to speed up parameter tuning, as well as an adaption of the GeneMania network combination algorithm that takes advantage of these specific negative examples. Finally, we introduce a new evaluation metric with useful interpretability for the function prediction problem, and demonstrate improved accuracy of our algorithm over existing state of the art in this and other metrics. On all metrics are algorithm reduces error by between x and y percent.

**Availability:** Code and Data is available at your mom’s house

**1. Introduction**

Since the MouseFunc competition, published in 2008 [Pena Castillo 2008], there has been a surge of interest in applying traditional machine learning methods to the problem of protein function prediction (PFP). The PFP problem had already been highlighted as one of importance to biologists, since the biological function of a significant portion of sequenced genes have unknown function. Traditionally approaches to PFP involved either homology (with limitations of accuracy) or curation(dependent on rare expertise). Yet there is an ever-increasing amount of biological data beyond gene sequence data that provides insight into protein function, such as tertiary structure (cite HPF). The MouseFunc competition showed that traditional machine learning methods are capable of taking advantage of such data to provide useful predictions of gene function.

The methods utilized by MouseFunc competitors were quite diverse: support vector machines, Random Forests, Decision Trees, etc. [cite these], but one recurring theme that has since been repeated in several variations in the field was the idea of guilt-by-association. 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 principal component of feature vectors). One of the most elegant and best-performing of these methods, is the GeneMania algorithm [Mostafavi 2008], which incorporates prior beliefs and an intelligent network combination algorithm into its guilt-by-association framework.

Here we focus on several of the challenges identified in the ongoing work on the GeneMania algorithm, which are also more broadly applicable to the PFP problem in general: Utilizing available data to form prior beliefs about the biological functions of a gene, choosing a set of high-confidence negative examples and assigning strengths to them, and combining disparate data sources. We present a parameterizable Bayesian technique for computing prior functional biases for each gene, as well as a framework for tuning these as well as the other parameters in the original GeneMania algorithm. To facilitate this parameter tuning, we incorporate new optimization techniques that significantly reduce computation time. We also present a novel way of identifying negative examples that utilizes our computed priors and integrate these negative examples into the network combination algorithm.

Our algorithm demonstrates significant improvement in predictions in several different evaluation settings, as well as reduced computation time. We also believe that along with enhanced performance, our algorithm exhibits knowledge-based tendencies that augment its understandability, and position it to gain further performance advantages as new techniques for choosing negative examples are developed. Lastly, we discuss the pitfalls of the evaluation of PFP methods with traditional machine learning statistics, and present a new evaluation statistic that is more easily interpretable in the context of predictions for experimental biologists to evaluate.

**2. Previous Work**

As mentioned, our work is based on the GeneMania function prediction algorithm of Mostafavi 2008, one of the top performing competitors in the MouseFunc competition (Pena-Castillo 2008). GeneMania is a form of Gaussian Random Field 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 y of prior label biases for the nodes given the current function being examined, the GRF algorithm assigns a discriminant value [should identify this with f at this point; for every equation please make sure to define the terms before the equation appears or just after] to each node, which can be ranked in order to produce predictions. The label biases yi take values in [-1,1], with -1 representing known negative labels and 1 representing known positive labels. The final discriminant values, fi, are obtained by solving the optimization problem:

 eq 1.

Whose analytical solution is: (I + L)f = y, with L = D – W, where D is a diagonal matrix with Dii = ∑k wik. Obtaining this solution involves only solving a linear system of the form Ax=b, and with proper normalization of W (the pairwise similarity matrix), A is guaranteed to be symmetric positive-definite. Thus the conjugate gradient algorithm can speedily and reliably solve for the discriminant vector 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, 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. 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 2009 explores variations on techniques to choose the label bias vector, one of which we expand upon in our algorithm.

The other key component of the GRF algorithm is the composite network defining similarity between all pairs of genes. Mostafavi 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 2005. This algorithm, for each GO category of interest, maximizes the similarity between pairs or positively labeled genes and minimizes the similarity between genes of opposite labels. This is accomplished by calculating the final network W\* as the weighted sum of each individual network Wi, with the weights chosen to solve  eq 2.

If there are n nodes, pl positively labeled nodes, nl negatively labeled nodes, and h different data types, then **Ω** is a (pl^2 + pl\*nl) by h [use exponents instead of ^2] matrix, where each column contains all of the entries in Wi corresponding to the positive-positive and positive-negative label pairs, and **t** is a target vector taking on the values (nl / n)^2 for positive-positive pairs and (-nl\*pl) / (n^2) for positive-negative pairs.

This network combination algorithm becomes difficult in cases with few positive examples, and in general is prone to over-fitting. Mostafavi 2008 originally combats this problem by introducing a regularization term, but in later work (Mostafavi 2010) instead attempts to fit the composite data network for multiple GO categories simultaneously. Our algorithm expands upon this second approach.

**3. Algorithm**

We propose novel techniques focusing on several key aspects of GRF-based protein function prediction: choosing negative examples, forming label biases for unlabeled genes with some known annotations, and combining data networks. In addition we suggest a new optimization algorithm tailored to our techniques, and provide a framework for tuning parameters using the training data. [This last sentence is not really the algorithm so should be eliminated] Lastly we explore different error metrics commonly used to judge machine learning algorithms, and focus on error metrics particularly useful to the experimentalists who would benefit most from computational predictions.

**3.1 Label Biases**

Mostafavi 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. This idea is supported by the work of King 2003, which showed that patterns of GO annotations alone provided enough signal to predict future annotations. The implementation of Mostafavi 2009, denoted HLBias, specified that genes which possessed annotations for functions ancestral to the function of interest received a prior bias of the proportion of genes with the ancestral function that also are known to have the function in question. [I don’t understand the following sentence.] However, the complexity of functional relationships, and the subjectivity of their delineation, suggest linkage between functions beyond ancestral relationships. 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, in the following manner:

Let G be the n by d annotation matrix, where n is the number of genes and d the number of GO categories, and let P be the d by d matrix specifying the empirical conditional probability of seeing annotation c given the presence of annotation m [This notation is strange: P(m,c) is the notation for a joint probability but what you say in words is P(c|m)]:

P(m,c) = p(Gic=1 | Gim=1) = n+mc / n+m

For a protein i with annotations dj in all GO categories across all three branches, we approximate the conditional prior probability of gene i having function c by the score:



[this formula says that if gene i has many known go categories, then it is less likely to have this one] This is efficiently calculable for all genes and all functions simultaneously as:

Y = wGP

Where , and each of the resulting columns of Y is the vector of label biases for GO category c. When calculating these label biases, all training annotations are used, even those for categories that fall outside of the 3-300 count restriction used to determine which categories to predict.

Due to the hierarchical nature of GO categories, some of the conditional probabilities in this calculation will contain redundant information **(Do I need to provide an example of this?) [Dennis doesn’t think so]**, and so we calculate biases as Y = wG’P, where G’ is an annotation matrix containing only leaf categories: G’ic=0 if Gic = 0 or if there exists m s.t. Gim = 1 and m is a child of c.

Lastly, we observe a large prevalence of score contamination from categories with small sample size, where one category appears to be a perfect predictor of another. In order to reduce the noise from this phenomenon, we introduce a weighted pseudocount into the calculation of the matrix P, whereby P(m,c) = n+mc / n+m is replaced by:

P’(m,c) = n+mc / (n+m + **γ\*** exp(λ \* n+m))

There are two assumptions behind this parametric pseudocounting procedure: no category co-occurs with a non-ancestral category in every true case [what does this last phrase mean?], and the number of remaining but currently unknown occurrences of a function is related to the number of currently known occurrences. There are two competing hypotheses in regards to this latter idea: The first is that 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. The second hypothesis is that the number of currently known occurrences is in fact a better representation of the specificity of a function, and so the undiscovered occurrences should only be a function of the number of genes left to study, in which case the number of missing counts in the data 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 **γ** in [0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512]

and λ in [-0.05, -0.025, -0.0125, -0.00625, -0.003125, 0, 0.003125, 0.00625, 0.0125, 0.025, 0.05]. Including λ = 0 in the range allows for the potential for a constant pseudocount. The final value of the parameters is chosen by tuning with cross-validation over the training set, as described in section 3.5.

Lastly, for genes with no previous annotations in GO, we follow Mostafavi 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 as ALBias.

**3.2 Negative examples**

The choice of negative examples for use in supervised machine learning algorithms is a recurring problem for the protein function prediction algorithm. While the GO ontology does include negative annotations, the number of such annotations is currently small. Thus it becomes necessary to infer negative examples for each function using some heuristic. Past examples of this have included merely assuming all non-positive genes with any other annotation are negative, randomly sampling genes and assuming the probability of getting a false negative is low, and using genes with annotations in sibling categories of the category of interest as negative examples. Mostafavi 2009 uses this last technique, but notes in discussion that this choice may often break down as many genes are annotated to more than one sibling category.

We present a new technique for choosing negative examples based upon the label biases calculated for each function. Namely, all genes with any annotation in any of the three branches of GO (including those outside of the usual 3-300 specificity range), but which have a bias score of 0 for the function in question, are treated as negative examples. Intuitively, this amounts to choosing genes as negative examples which have some annotations, but whose annotations have never appeared alongside the annotation in question in any gene. It is important to 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 or not a bias is non-zero. [Dennis has been changing choosing negative examples to treating negative examples, because he thinks that negative examples may have different values of negativity. If not, then choosing is better than treating.]

**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 2010 found that fitting this matrix for multiple GO functions simultaneously significantly decreased over-fitting, especially in low-annotation categories. This simultaneous fit, called Simultaneous Weights (SW), is obtained by solving the equation:

  eq 3

[The following discussion can go in the supplement since you don’t use this formulation:] Where each c is a different GO category in the same branch of the hierarchy. This equation is a simplification of the formulation:

 eq 4

This simplification is made possible by considering negative-negative pairs of labels as well as the positive-positive and positive-negative, and treating all non-positive nodes as negative nodes, which causes all **Ωc** to be identical, and all tc vectors to be the same length.

Mostafavi 2010 showed that these simplifications do not hamper performance, and also founde that fitting the combined network to all GO categories in a particular branch worked better than any other subset or grouping of functions. We concur that fitting to all categories performs better than any of the subsets we attempted, but propose that lack of performance degradation when assuming that all non-positive nodes were negative was most likely due to a lack of any satisfactory alternative for choosing negative examples.

[this part should be in the paper:] We return to the formulation: 

But maintain the unique **Ωc** matrices dependent upon both the positive labels and the specific negative examples chosen for category c. Such a formulation still has an analytical solution:, which can be efficiently solved with a Cholesky decomposition, as [formatting messed up]  is positive definite and is only h x h in dimension with h being the number of data sets being combined. We refer to our network algorithm as Simultaneous Weights with Specific Negatives (SWSN), and note that while the SW algorithm is appropriate for all categories with less than 300 annotations, we fit our SWSN algorithm only to the set of categories where function prediction is to be performed (between 400-500 categories for each benchmark). Our experiments indicate that this reduction in the number of categories fit has a negligible impact on performance.

**3.4 Successive Block Conjugate Gradient Optimization**

The network-weighting scheme defined above creates a single combined matrix W for all functional categories within the same GO branch. Examining the optimization problem that needs to be solved to obtain discriminant values for each function, we observe that the coefficient matrix is identical across functions, and so we are faced only with the issue of differing Right-Hand-Sides. 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 of Suarjana and Law (1994).

In this algorithm, which we refer to as SBCG, the search direction is obtained simultaneously for all of the distinct right hand side (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 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.

[supplement:] Our algorithm differs from the original proposed by Suarjana and Law in several small but useful ways. Firstly, not all solutions converge to the desired tolerance in the same number of iterations, and so we can 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 dangerously 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 split up the function prediction problems into with a maximum of 500 RHS vectors

**Algorithm 1**

**….**

**3.5 Parameter Tuning**

The multiple RHS framework described in 3.4 lends itself well to parameter tuning, as the different combinations of the parameters λ and **γ** described in 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 2004 included an additional parameter μ, which describes the relative weight to be placed on each component of the objective function, which we formulate as:



This parameter was ignored by all previous versions of GeneMania, but we test its impact on function prediction by adding it to our tuning methodology. After obtaining the best parameters λ\* and **γ\*,** we tune for μ as well, using λ\* and **γ\*** for the ALBias computation in course of tuning. [What is the justification for this sequential approach as opposed to optimizing all three parameters simultaneously?]

In order to choose these performance-maximizing parameters, we subdivide the training data into a tuning subset and a validation subset, with sizes of 2/3 and 1/3 of the training data respectively. We also take care to ensure that the proportion of genes with any GO annotations to those that are completely unannotated are the same in each subset.

When tuning in the novel setting, the label biases rely on existing annotations, and so we choose not to wipe all annotations in the validation subset, but rather to attempt to re-create a novel-like setting. We accomplish this with algorithm 2, which removes a random subset of annotations from a smaller subset of genes in the validation subset, as well as completely wiping 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 data. Any categories where no annotations were removed from the validation subset are deleted from the list of categories to be predicted.

Once predictions are made for the validation subset, evaluation scores are calculated for the removed annotations in the validation subset. In order to choose parameters that simultaneously maximize all scores, we combine them into a single scoring metric, but first normalize each score across all parameter combinations in the standard fashion: SCORE\_NORM = (SCORE – mean(SCORE)) / std(SCORE). This normalization step addresses the issue that the same magnitude of change in different score metrics has different meaning (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, as discussed in section 4.3). We define our parameter score as the following combination of metrics:

PARAM\_SCORE = AUC\_ROC\_NORM + AUC\_PR\_NORM + .5\*TOP10\_NORM + .3333\*TOP100\_NORM + .1666\*TOP1000\_NORM, and choose the parameters that maximize this score (TOP scores are defined in section 4.4). [What is the justification for these magic numbers?]

When applying the SBCG algorithm, we find a large amount of rank deficiency amongst [amongst is archaic; could you use 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 most likely be to solve all blocks simultaneously in one large system, but we did not explore this option due to memory constraints.

**Algorithm 2**

**…**

**4 Methods**

**4.1 Evaluation datasets**

We evaluate our algorithm on the MouseFunc benchmark, focusing on the Molecular Function branch of the GO hierarchy. This data includes 10 networks (Interpro data, PFAM data, 3 Gene Expression networks, PPI data, Phenotype, 2 Conservation Profile networks, Disease Association data), 1874 molecular function categories(spanning all annotation count ranges) and 21603 mouse genes, with all data gathered in 2006.

We also evaluate performance on the Biological Process branch of the GO ontology in Yeast, using data obtained from Mostafavi 2010, which includes 44 networks of data obtained from BIOGRID (Stark 2006), covering 3904 genes with 1188 biological process categories (which are all categories with between 3 and 300 annotations). We augment this yeast data with experimentally confirmed gold-standard annotations in the BP category of GO:0007005, mitochondrion organization and biogenesis (MOB), obtained from Huttenhower 2009.

**4.1 Functional association data**

Association networks are created from feature-based data types using the PCC [spell this out], after a frequency transform as described in Mostafavi 2008. Only the top 100 interactions are used for each gene in the training set in order to keep the networks sparse, and a normalization scheme of W’h = Dh1/2WhDh1/2 is applied to each network and the final combined network, where Dh is the diagonal row sum of Wh.

**4.2 Evaluation Frameworks**

For the purposes of evaluation, protein functions are described by GO ontology annotations, and predictions are made only for those functions annotated to between 3 and 300 gene products [is gene product the same as a gene? Also how would we know which are the appropriate annotations a priori?] in each genome. When considering functional annotations, the common convention of excluding ‘IEA’ annotations is observed.

As in the MouseFunc competition, performance is evaluated in two different scenarios: a test set where a subset of GO-annotated data (1718 genes in mouse) is removed from all annotations and then predictions are made from the remaining training data, and 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 categories we are attempting predictions for (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.

[supplement] Pena-Castillo 2008 remarks that there appears to be a qualitative difference between the two types of evaluations, and indeed the performance of all algorithms is markedly higher on the test set than on the novel set. This dichotomy in performance is mirrored in later work by Mostafavi 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 GO 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 in order 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 novel evaluation setting.

The second possible factor lies in the interdependence of both the data and the annotations on homology modeling. Several of the included data-types: Pfam, Interpro, OMIM, etc. use homology modeling 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 wiped out for the test set, the homology links that caused 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 setting, 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 homology modeling not reflected in the training data.

We treat the novel set as the more important evaluation scheme for our algorithmic changes, as in our opinion it reflects a more useful scenario, and suffers from fewer potential biases. 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 categories as the original competition: 488 and 442 categories respectively. For yeast, we show results only the novel setting, with data from June 2007, one year after the training data, which includes 511 GO BP categories with at least one new annotation.

It also bears mentioning that 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 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 setting 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 the “Golden Set” benchmark of yeast genes experimentally verified by Huttenhower 2009 for GO:000705 (MOB). These golden set 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 golden set, we add an additional 2473 genes to the 342 comprising the yeast novel set, which come from negative examples from Huttenhower 2009 that are present in our gene set.

**4.3 PR vs. ROC**

The performance of discriminant-based classification algorithms is most often represented by two plots: The ROC curve, which plots (true positives) / (true positives + false negatives) as a function of (false positives) / (false positives + true negatives), and the PR curve, which plots (true positives) / (true positives + false positives) as a function of (true positives) / (true positives + false negatives). Each of these curves can be summarized by their AUC, the area that the curve encompasses. For ROC, this area has a nice interpretation: the probability that a randomly chosen true positive will be ranked higher by the classifier than a randomly chosen true negative. For PR, no such neat interpretation exists, but the area does represent how close the classifier is to a perfect oracle, which would predict no false positives and have an AUC of 1.

While both performance measures attempt to describe how well the ordering of discriminant values captures the true positive and negative labels, each tends to reward slightly different behavior. AUC\_PR has the largest marginal difference at the top of the ranking range, where a move from rank 1 to rank 2, for example, causes a drop of 50% in precision, while the relative impact of a downward move in rank decreases exponentially farther lower on the list. Conversely, AUC\_ROC moves linearly with list ordering, and so in data sets with large skew, differences between the highest orderings are quite small. As a result, AUC\_PR will reward a combination of excellent and poor ranking, while AUC\_ROC would prefer mediocre ranking across all labels, as shown in figure 1.a.

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 1.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 categories. Figure 1.c illustrates such a case, where large improvement in one poor classifier is drowned out by a small decrease in performance of an excellent classifier.

**4.4 TOP\_SCORE**

Freitas 2010 points out the importance of confidence if computational predictions are to be useful. An experimentalist must be able to choose how many predictions to attempt to assay given limited resources, and also which functions to focus on, based on classifier performance.

In order to create a metric more robust to averaging than PR, but which still has easy interpretability for experimentalists, we propose TOP\_SCOREC, defined as: [(True positives < rank C) / C ] / [min(C, positive label count) / C]. 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 TOP\_SCORE10, TOP\_SCORE100, and TOP\_SCORE1000 for mouse, and TOP\_SCORE10, TOP\_SCORE50, and TOP\_SCORE200 for yeast, providing insight into the usefulness of computational predictions at three different scales of experimental testing. Table 1.d illustrates the differences in TOP\_SCORE for the same scenario as Table 1.d

**Table 1**

|  |  |  |  |
| --- | --- | --- | --- |
| a. AUC\_PR and AUC\_ROC scores for excellent-poor and mediocre-mediocre rankings | b. High AUC\_ROC due to large skew, in spite of a poor prediction | c. Average AUC\_PR for an excellent and poor classifier, and decrease in Average AUC\_PR in spite of vast improvement in the poor classifier | d. Average TOP\_SCORE values for an excellent and poor classifier, and increase in score after a vast improvement in the poor classifier |
| 2 true positive, 1998 true negatives  Classifier gives the true positives ranks of 1 and 500, AUC\_ROC= 0.8754  AUC\_PR = 0.7510  Classifier gives the true positives ranks of 20 and 21, AUC\_ROC= 0.9905  AUC\_PR = 0.0613 | 1 true positive, 9999 true negatives  Classifier gives the true positive a rank of 500, AUC\_ROC= 0.9501  AUC\_PR= 0.0020 | 2 true positives, 5998 true negatives  3 true positives, 5997 true negatives  Classifier A gives the true positives ranks of 3 and 12  Classifier B gives the true positives ranks of 500, 600, and 1100  Average AUC\_PR = 0.1471  Classifier A gives the true positives ranks of 4 and 11  Classifier B gives the true positives ranks of 50, 60, and 110  Average AUC\_PR:  0.1293 | 2 true positives, 5998 true negatives  3 true positives, 5997 true negatives  Classifier A gives the true positives ranks of 3 and 12  Classifier B gives the true positives ranks of 500, 600, and 1100  TOP\_SCORE10 =.25  TOP\_SCORE100 =.50  TOP\_SCORE1000 =.833  Classifier A gives the true positives ranks of 4 and 11  Classifier B gives the true positives ranks of 50, 60, and 110  TOP\_SCORE10 =.25  TOP\_SCORE100 =.833  TOP\_SCORE1000 =1 |

**5. Results And Discussion**

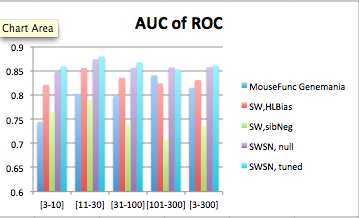
We present results for our proposed techniques as per the discussion of evaluation metrics and datasets in section 4. These results are divided according to their impact on computational or performance metrics, as well as a section devoted to the outcomes of the parameter tuning.

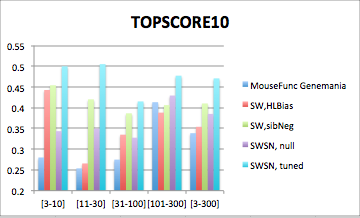
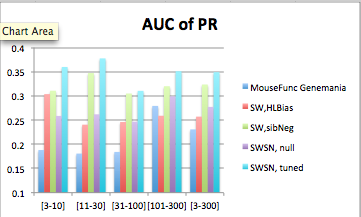
**5.1 Prediction Evaluations**

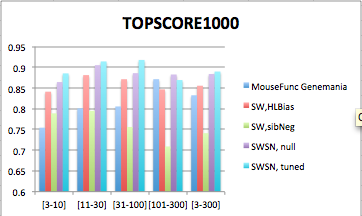
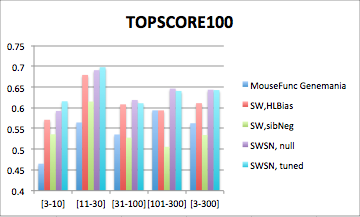
We present the performance, evaluated by AUC\_ROC, AUC\_PR, and TOP\_SCORE metrics, of five algorithms: the original MouseFun GeneMania algorithm, the FastSW GeneMania algorithm presented in Mostafavi 2010 using sibling negative examples,the FastSW algorithm combined with the HLBias algorithm of Mostafavi 2009, and two versions of our algorithm, SWSN with ALBias and null parameters (λ =0, **γ** =0, μ=0.5), and SWSN with tuned parameters.

In the novel setting for MouseFunc, our algorithms show a strong increase in performance across all metrics, especially our version with tuned parameters.

**Figure 1. Performance metrics in the novel setting in mouse**

****

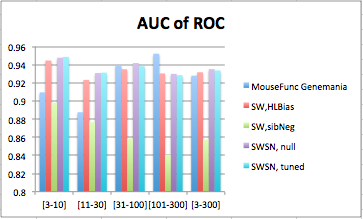
****

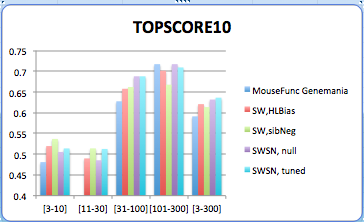
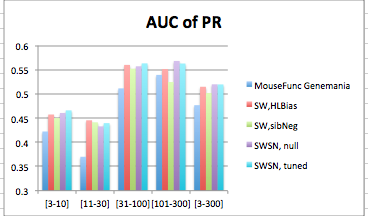
****

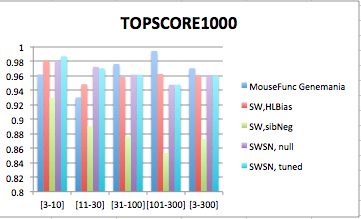
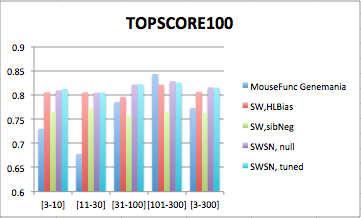
For the mouse test set, the difference in performance is much smaller, as there are no labels to use in ALBias for the test genes. Yet we still see a performance increase in our algorithms across nearly all metrics, due to better biases for genes with

edges connected to test genes.

**Figure 2. Performance metrics in the test setting in mouse.**

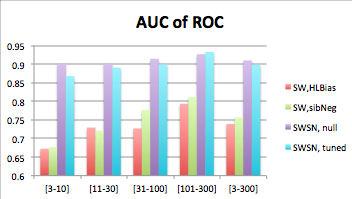
****

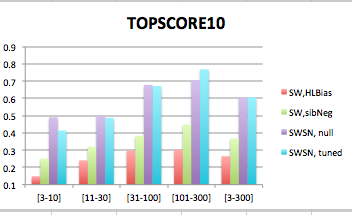
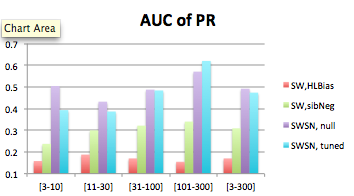
****

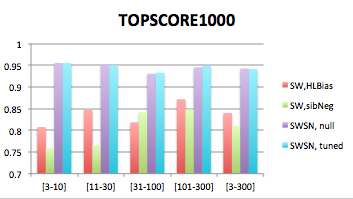
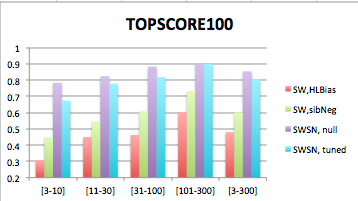
****

In the yeast novel set we compare the same algorithms minus the original MouseFunc GeneMania algorithm, and observe dramatic performance increase of our algorithms across all evaluation metrics.

**Figure 3. Performance metrics in the novel setting in yeast**

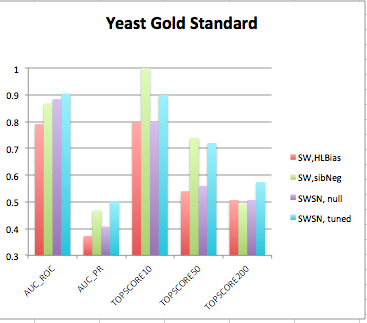






Lastly on the yeast MOB Golden Set, we see strong performance from our tuned SWSN ALBias algorithm, which achieved significantly higher AUC\_ROC, AUC\_PR, and TOPSCORE\_200 scores, but also from the SW, sibNeg algorithm, which achieved the highest TOPSCORE\_10 and TOPSCORE\_50 scores.

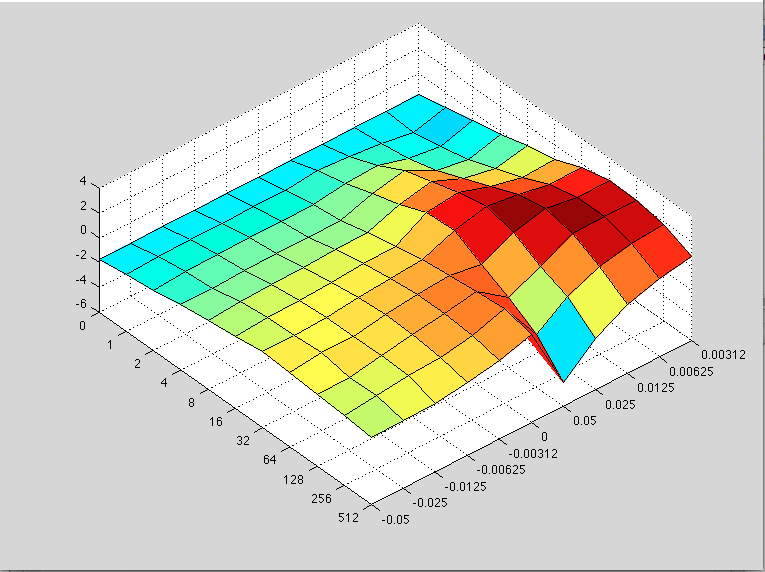
**Figure 4. Performance metrics on the yeast gold standard**

****

**5.2 Parameter Tuning Results**

**[This whole section should go in the supplement with just a sentence saying that the algorithm is insensitive to parameter selection]** As described in section 3.5, the parameters are chosen via a tuning process on the training data that computes a combined score for all pairs of candidate parameters λ and **γ**, and then subsequently tunes for μ. A sample of these scores for the novel mouse setting is depicted in figure 5. 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. [Nice to compare this approach with a simultaneous tuning of all parameters]

**Figure 5. Parameter tuning scores**



The best parameters resulting from the tuning process in each setting are listed in table 2, and the positive values for **γ**are evidence for the 2nd hypothesis put forth in section 3.1, 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 the mouse novel setting, where the information contained in the association network was more important than restricting genes to their prior biases.

**Table 2. Tuned parameters for each evaluation benchmark**

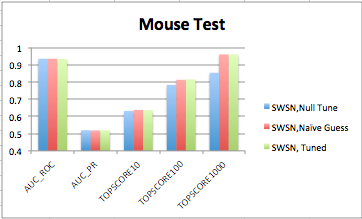
|  |  |  |  |
| --- | --- | --- | --- |
|  | λ | **γ** | μ |
| Mouse Novel | 64 | 0.0125 | 0.2 |
| Mouse Test | 32 | 0.0125 | 0.5 |
| Yeast Novel (and MOB Golden Set) | 32 | 0.025 | 0.4 |

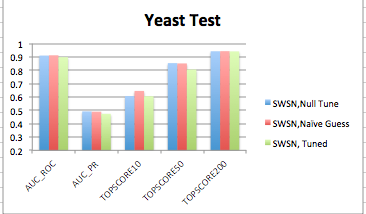
From the performance measurements presented in section 5.1, we see that while the tuned parameters performed significantly better than the null parameters in the mouse novel and yeast MOB benchmarks, their performance was on par with, or occasionally worse than the null guess in the mouse test and yeast novel benchmarks.

Further investigation shows that a naïve guess of the parameters (λ =16, **γ** =-.0125, μ=0.4) performs as well or better than the tuned parameters in most metrics in our evaluation categories, as shown in in figure 6.

**Figure 6. Performance metrics for tuned and unturned parameters**

****

****

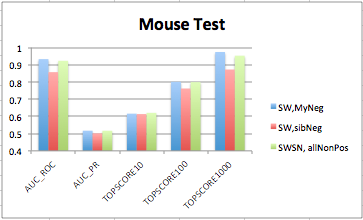
****

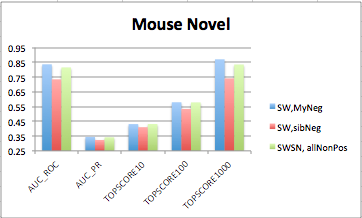
We attribute this difficulty in tuning parameters to the fact that despite our efforts with algorithm 2, our synthetic novel tuning set still suffers from many of the biases in cross-validation testing discussed in section 4.2. One possible remedy to this problem would be to tune the parameters with 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. There is also the possibility of tuning all three parameters at once, rather than in two different iterations. Lastly, we see the need for a different methodology to deal with the differing magnitude of score differences, other than the normalization scheme proposed, as our scheme is sensitive to the candidate parameters chosen, where a parameter combination that performs very poorly in only one of the evaluation metrics will skew the magnitude of the contribution of that metric for all the strong performers in that metric **(I need to find a better way to say that)**.

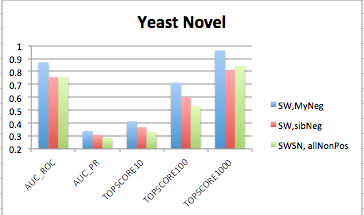
**5.3 Negative Example Choice**

One of the largest factors contributing to the success of our algorithm is our choice of negative examples. We investigate this further by comparing the SW network combination algorithm with no label bias method, and three different negative example methods: the sibling negative examples, setting all non positive genes with GO annotations to negative examples, and our new negative example approach. As shown in figure 7, in all cases our negative example choice outperforms previous choices, and approaches the performance of our full SWSN with ALBias and tuned parameters algorithm in several metrics.

**Figure 7. Performance metrics for negative example choices**

****

****

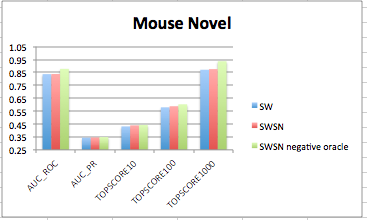
****

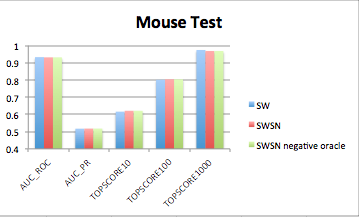
Despite the success of our method, in some GO categories there are still several instances of genes selected as negative examples that ended up with a positive annotation in the novel set. Accordingly, we believe there is more potential for refined methods that correctly define high-confidence negative examples, and that these methods will have high impact in the performance of machine learning algorithms. [supplement:] Indeed, we hypothesize that part of the performance gain demonstrated by the HLBias algorithm in Mostafavi 2009, was due to the fact that they adjusted the labels for all non-positive genes, effectively turning any gene without a label in an ancestral category of the function in question into a negative example. **(I can give some evidence for this hypothesis if necessary, with another graph or table, by showing the downgrade in performance when you don’t adjust all non-pos with go)**

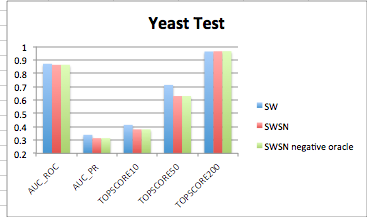
**5.4 Network Combination Algorithm SWSN**

**[If this is true, then Dennis suggests presenting only SW. Simpler is better and more general.]** A comparison of SWSN to the SW network weight algorithm, using no label biases and our negative examples, yields mixed results across evaluation settings and metrics. In spite of this we believe that our method has a greater intuitive backing, and that further refinement of negative example choice will allow our method to outperform the SW method. To demonstrate this, we add to the comparison of the two algorithms in figure 8, a third algorithm in which our negative examples are granted access to the novel evaluation set, 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 novel set). This results in strong performance on the mouse novel set, but makes no difference on the mouse test or mouse novel sets, as there were no instances of negative examples that were demonstrated positives in mouse test and only 11 in the yeast novel benchmark.

**Figure 8. Performance metrics for network combination algorithms**

****

****

****

**4.2 Computational Cost**

**[supplment]** 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(d\*n^2) per iteration where d is the total number of GO categories to be predicted. The per-iteration cost of our Successive Block Conjugate Gradient Variant is O(d\*n^2 + d^2\*n + d^3) (see supplement for complexity analysis). 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 categories for the novel evaluation

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 in order to measure algorithmic performance. Table 1 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 SBCGV algorithm, for different numbers of categories, as well as the final norm of the residual matrix: norm(Ax-b).

These results show a 30% reduction in the number of flops required for the prediction task on original MouseFunc data, while converging to a solution with smaller residuals as well. On the yeast benchmark, the reduction in flops was lower at 22% due to the fact that the ration between n and d is much smaller. As expected there is an observable increase in computation saved as the number of categories increases, but this is bounded by the fact that our algorithm splits the categories into subsets with a maximum size of 500. This suggests that further computation could be saved by devising a suitable strategy to deal with unruly condition numbers for larger sets of RHS vectors. The work of Suarjana and Law also suggests that a pre-conditioner applied to the data might help reduce the number of iterations required as well.

**Table 3 flops and error defined as the norm of the residual matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **Setting** | **NO. of categories** | **Serial Conj Grad**  **(Flops, error)** | **SBCGV**  **(Flops, error)** |
| **Mouse Example** | **10** | **(1.8206e+11, 1.4869e-08)** | **(1.5343e+11, 1.1381e-08)** |
| **Mouse parameter tuning** | **342** | **(5.8418e+12, 6.1187e-08)** | **(4.0961e+12, 5.8895e-08)** |
| **Mouse parameter tuning, 4 pairs of parameters** | **1420 (subsets of 474, 474, and 472)** | **(2.4178e+13, 1.1884e-07)** | **(1.6844e+13, 9.1782e-08)** |
| **Yeast parameter tuning** | **426** | **(1.9368e+11, 3.3293e-08)** | **(1.5216e+11, 3.7322e-08)** |
| **Yeast parameter tuning, 4 pairs of parameters** | **1704**  **(subsets of 426)** | **(7.9071e+11, 8.8582e-08)** | **(6.1603e+11, 5.5898e-08)** |

**5. Conclusion**

We presented several adaptations of the GeneMania function prediction algorithm, which resulted in substantial performance increases across several benchmarks. These improvements included a novel method of choosing negative examples, which is broadly applicable to other function prediction methods as well, and a parameterized Bayesian methodology for computing prior label biases. We also introduced a new optimization methodology, which substantially decreased computational costs.

We devised a framework for tuning parameters in a synthetic novel set which added further performance gain in the novel setting in mouse, but requires additional work to combat biases inherent in the association networks. Our new SWSN network combination algorithm shows promise if even-better performing negative example methods are developed, though it does not significantly improve performance as of now. Finally, we presented a new evaluation metric that will help experimentalists choose predictions to experimentally verify while still being robust to averaging over multiple function categories.

When comparing performance statistics of different algorithms, it is important to keep in mind that a difference of a few percentage points can mean hundreds of new true annotations when applied across all functions. 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..

In summary we have shown that function prediction through guilt-by-association can be computed with even more accuracy and efficiency than ever before, and provided insight into some of the inherent difficulties still facing the development and evaluation of protein function prediction algorithms.

**7. References**

Couldn’t have done it without