BIOINFORMATICS
Vol. 00 no. 00 2005
Pages ???

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

Noah Youngs [1], Duncan Penfold-Brown [2], Kevin Drew [2], Dennis Shasha [1,*], and Rich Bonneau [1,2 *]

[1]Department of Computer Science, New York University, New York, USA
[2]Center for Genomics and Systems Biology, Department of Biology, New York University, New York, USA

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

1 SUPPLEMENT

GeneMANIA Label Propagation Solution

In Gaussian Random Field label propagation, the discriminant values defining the ranking of predictions are obtained by solving an optimization problem which seeks to let prior label information flow throughout the network, mitigated by the strengths of the edges in that network, while at the same time constraining the labels of nodes to remain close to their prior value. The final discriminant vector $\vec{f}$ is obtained by solving the optimization problem:

$$\min_f \left(\sum (f_i - y_i)^2 + \sum \sum W_{ij}(f_i - f_j)^2\right) \tag{1}$$

Whose analytical solution is obtained by solving:

$$(I + L)\vec{f} = \vec{y} \tag{2}$$

With $L = D - W$, 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 $D_{ii} = \sum_j W_{ij}$. Obtaining this solution only requires solving a linear system of the form $Ax = b$, 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 $\vec{f}$.

GeneMANIA Network Combination

The original GeneMANIA algorithm (Mostafavi *et al.*, 2008) solved the problem of combining disparate data similarity networks through a data-driven optimization that 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 W^*as the weighted sum of each individual network W_i, with the weights chosen to solve:

$$\vec{\alpha}^* = \min_{\vec{\alpha}} (\Omega\vec{\alpha} - \vec{t})^{\intercal}(\Omega\vec{\alpha} - \vec{t}) \tag{3}$$

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 matrix, where each column contains all of the entries in W_i corresponding to the positive-positive and positive-negative label pairs. The target vector $\vec{t}$ contains the values: $\frac{nl}{n}^2$ for positive-positive pairs and $\frac{-nlpl}{n^2}$ for positive-negative pairs, in order to deal with class imbalance in the labeled data. The resulting vector $\vec{\alpha}^*$ 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.

Parameter Tuning Ranges

We present a pseudocounting procedure for our Bayesian functional bias method in section 3.1 of the main text. This procedure addresses the issue of low sample size that occurs in many functions, and expresses the hypothesis that the number of undiscovered occurrences of a function is related to the number that are currently annotated. In order to tune our pseudocounting procedure for each evaluation scenario, we sample from a range of parameters: $\gamma \in$ [0, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512] and $\lambda \in$ [-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:

$$\hat{p'}(c|m) = \frac{n^+_{mc}}{n^+_m + \gamma e^{\lambda n^+_m}} \tag{4}$$

Including $\lambda = 0$ in the range allows for the potential for a constant pseudocount (or no pseudocount at all if $\gamma = 0$). We also test values of μ in [.05, .1, .15, ..., .9, .95], where μ is the parameter guiding the relative importance of the smoothness and consistency terms in the GRF objective function (see section 3.5 of the main text):

$$\min_f \mu \sum (f_i - y_i)^2 + (1 - \mu) \sum \sum w_{ij}(f_i - f_j)^2) \tag{5}$$

Network Weighting

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

$$\vec{\alpha}^* = \min_{\vec{\alpha}} -2\vec{\alpha}^{\intercal}\Omega\vec{t}^{\intercal} + h\vec{\alpha}^{\intercal}\Omega^{\intercal}\Omega\vec{\alpha} \tag{6}$$

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

$$\vec{\alpha}^* = \min_{\vec{\alpha}} \sum_{c=1}^{h} (\vec{t_c} - \Omega_c\vec{\alpha})^T(\vec{t_c} - \Omega_c\vec{\alpha}) \tag{7}$$

*to whom correspondence should be addressed

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 $\vec{t_c}$ vectors to be the same length.

As mentioned in the main text, Mostafavi and Morris (2010) showed these simplifications do not hamper performance, and also showed a better fit to all GO categories in a particular branch (GO-BP, GO-CC, or GO-MF) than any other subset or grouping of GO categories.

Our modification to the network combination algorithm relies on returning to the formulation of Equation (7) but also maintains the unique Ω_c matrices dependent upon both the positive labels and the specific negative examples chosen for category c. Such a formulation still has an analytical solution: $\sum_c \Omega_c^\intercal \vec{t_c} = (\sum_c \Omega_c^\intercal \Omega_c)\vec{\alpha^*}$, which can be efficiently solved with a Cholesky decomposition, as $(\sum_c \Omega_c^\intercal \Omega_c)$ is positive definite and is only h by 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 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 any of the performance metrics.

Successive Block Conjugate Gradient Algorithm

After observing the repetitive nature of the optimization problems required to solve Equation (1) in the main text, which must be solved for each function, we propose the use of a new class of optimization technique which deals with sets of linear equations with differing right-hand-sides.

The Successive Block Conjugate Gradient (SBCG) algorithm, originally proposed by Surajana and Law (1994), solves a set of optimization problems of the form $A_i x = b_i$, were all A_i are identical and only the b_i differ. This situation arises in GRF protein function prediction, as for each function we solve:

$$(I+L)\vec{f} = \vec{y} \tag{8}$$

With $L = D - W$, where I is the identity matrix, W the pairwise similarity matrix obtained by integrating multiple data types, and D is a diagonal matrix with $D_{ii} = \sum_j w_{ij}$. Since we compute W from all function categories at once, it is identical accross functions, and thus only the $\vec{y}$ vector differs, creating a situation suited to the SBCG algorithm.

The algorithm proceeds by computing a block search direction for all the residual vectors simultaneously. If at any point this search matrix becomes rank deficient, dependent residual vectors are moved to a secondary system, but still updated with information from the primary system, and thus continue to proceed to convergence. Accordingly, for each step of the algorithm computation is saved by sharing information across all of the right-hand -side problems.

Pseudocode for our adaptation of SBCG (adaptations described in section 3.4 of the main text) 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: $(\mathbf{R^m_{k-2}})^\intercal \mathbf{R^m_{k-2}} \beta = (\mathbf{R^m_{k-1}})^\intercal \mathbf{R^m_{k-1}}$
            $\mathbf{P_k} = \mathbf{R_{k-1}} + \mathbf{P_{k-1}} \beta$
        end if
        Orthonormalize $\mathbf{P_k}$, identify dependent indices d
          % Move dependent vectors to secondary system
        $\mathbf{R^m} = \mathbf{R^{m \setminus d}}, \quad \mathbf{R^s} = \mathbf{R^{s \cup d}}$
        $\mathbf{X^m} = \mathbf{X^{m \setminus d}}, \quad \mathbf{X^s} = \mathbf{X^{s \cup d}}$
          % Solve for search direction and steplength
        $\mathbf{U_k} = \mathbf{AP_k}$
        Solve: $\mathbf{P_k^\intercal U_k}[\alpha^\mathbf{m}_\mathbf{k}, \alpha^\mathbf{s}_\mathbf{k}] = [(\mathbf{R^m_{k-1}})^\intercal \mathbf{R^m_{k-1}}, (\mathbf{R^s_{k-1}})^\intercal \mathbf{R^m_{k-1}}]$
          % 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
                    $\mathbf{R^m} = \mathbf{R}^{\mathbf{m} \setminus i}, \quad \mathbf{R^c} = \mathbf{R}^{\mathbf{c} \cup i}$
                    $\mathbf{X^m} = \mathbf{X}^{\mathbf{m} \setminus i}, \quad \mathbf{X^c} = \mathbf{X}^{\mathbf{c} \cup i}$
                else
                    $\mathbf{R^s} = \mathbf{R}^{\mathbf{s} \setminus i}, \quad \mathbf{R^c} = \mathbf{R}^{\mathbf{c} \cup i}$
                    $\mathbf{X^s} = \mathbf{X}^{\mathbf{s} \setminus i}, \quad \mathbf{X^c} = \mathbf{X}^{\mathbf{c} \cup i}$
                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
```

Parameter Tuning

In order to tune the various parameters for our algorithm, we must create a subset within the training data in order to measure the performance of different parameter combinations (see section 3.5 of the main text). We begin by subdivide 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 unlabeled genes. 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 categories where no annotations were removed from the validation subset are deleted from the list of categories 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.

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 4.4 of the main text), 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 3.4 of the main text, 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.

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 $i = 1 \rightarrow \omega * |\Psi|$ 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 $i = 1 \rightarrow l$ 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 $\Phi$
```

Evaluation biases

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 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 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.

Evaluation Metrics

The two traditional measures of classifier performance are the ROC curve, which plots $\frac{\text{(true positives)}}{\text{(true positives + false negatives)}}$ as a function of $\frac{\text{(false positives)}}{\text{(false positives + true negatives)}}$, and the PR curve, which plots $\frac{\text{(true positives)}}{\text{(true positives + false positives)}}$ as a function of $\frac{\text{(true positives)}}{\text{(true positives + false negatives)}}$. Authors often present ROC performance in terms of the Area Under the Curve (AUC), and usually select a few fixed recall values for which to present precision statistics (although the AUC is also a valuable summary of the PR curve). For ROC, the AUC 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 S1.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 S1.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 S1.c and S1.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 (described in section 4.4 of the main text) preserves the interpretability of precision while alleviating some of the complications arising when averaging AUC_{PR} over multiple functions. Figure S1.c and S1.d illustrate the differences in TopScore alongside the average AUC_{PR}, showing that TopScore correctly captures the average improvement in classification.

Parameter Tuning Results

As described in section 3.5 of the main text, the parameters are chosen via a tuning process on the training data that computes a combined score for all combinations of candidate parameters λ, γ, and μ. 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 1, and the positive values for γ in the novel scenarios are evidence for the 2nd hypothesis put forth in section 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.

Table 1. Tuned parameters for each evaluation benchmark (see Equations 3 and 4 in the main text for parameter definition)

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

Further investigation shows that an observation-based guess of the parameters (λ=16, γ=-0.0125, α=0.4) performs competitively with the tuned parameters. Table 2, the results are shown for our ALBias algorithm with naive parameters (λ=0, γ=0, α=0.5), tuned parameters (values dependent on the evaluation scenario), and a set of guessed parameters (λ=16, γ=-0.0125, α=0.4). 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. As mentioned in the main text, the performance difference in yeast in particular is likely due to the fact that the novel tuning process attempts to recreate a situation of partial annotation, which is truer of a less-studied organism (mouse) than of a more studied organism (yeast).

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.

Evaluation Results

Figure S2 presents the same results as Figure 2 in the main text, but breaks down the results by the specificity of the function. The algorithms compared are 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 (λ=0, γ=0, α=0.5), and SWSN with ALBias and tuned parameters. The performance evaluations are averaged over functions within the same specificity buckets used

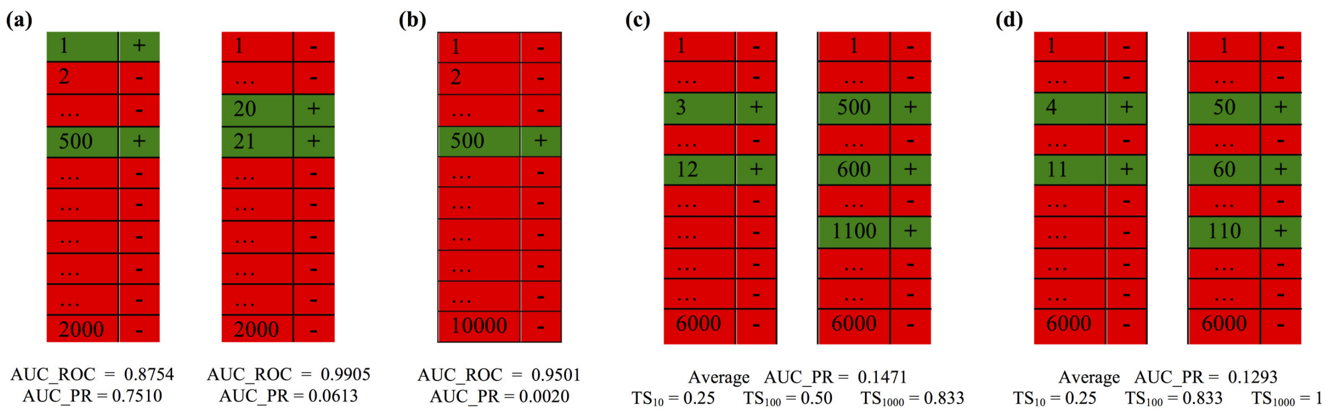

Figure S1 (a) AUC_{PR} and AUC_{ROC} scores for an excellent/poor and a mediocre/mediocre ranking of two true positives amidst 1,998 true negatives. (b) High AUC_{ROC} score of a poor ranking of one true positive amidst 9,999 true negatives. (c) Average AUC_{PR} and TopScore values for an excellent classifier 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 and a mediocre classifier of 3 true positives amongst 5,997 true negatives.

Table 2. Performance metrics for Naive Parameters (NP), Guessed Parameters (GP) and Tuned Parameters (TP)

Algorithm	AUC_{ROC}	AUC_{PR}	TS_{10}	TS^{*}_{100}	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}

in 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 S2.a. For the mouse test set, performance was strongest for the most specific categories, but suffered somewhat in more general categories when compared to the current GeneMANIA algorithm (Figure S2.b). In the yeast novel setting (Figure S2.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 presented in the main text.

Yeast Performance Improvement Examples

Examination of the GO terms for which our algorithm exhibited the greatest improvement in the yeast novel evaluation scenario, showed many occurrences of the label biases of validation true positives being improved by the use of GO annotations from branches other than Biological Process (the branch of the GO terms targeted for prediction). While it is likely that other changes in our algorithm contributed to the performance increase as well, the only observed systematic change in the predictions with most-improved metrics was the aforementioned bias improvement. One example of non-BP terms aiding in the prediction task was discussed in the main text, but we have included several more below to further demonstrate this occurrence.

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.

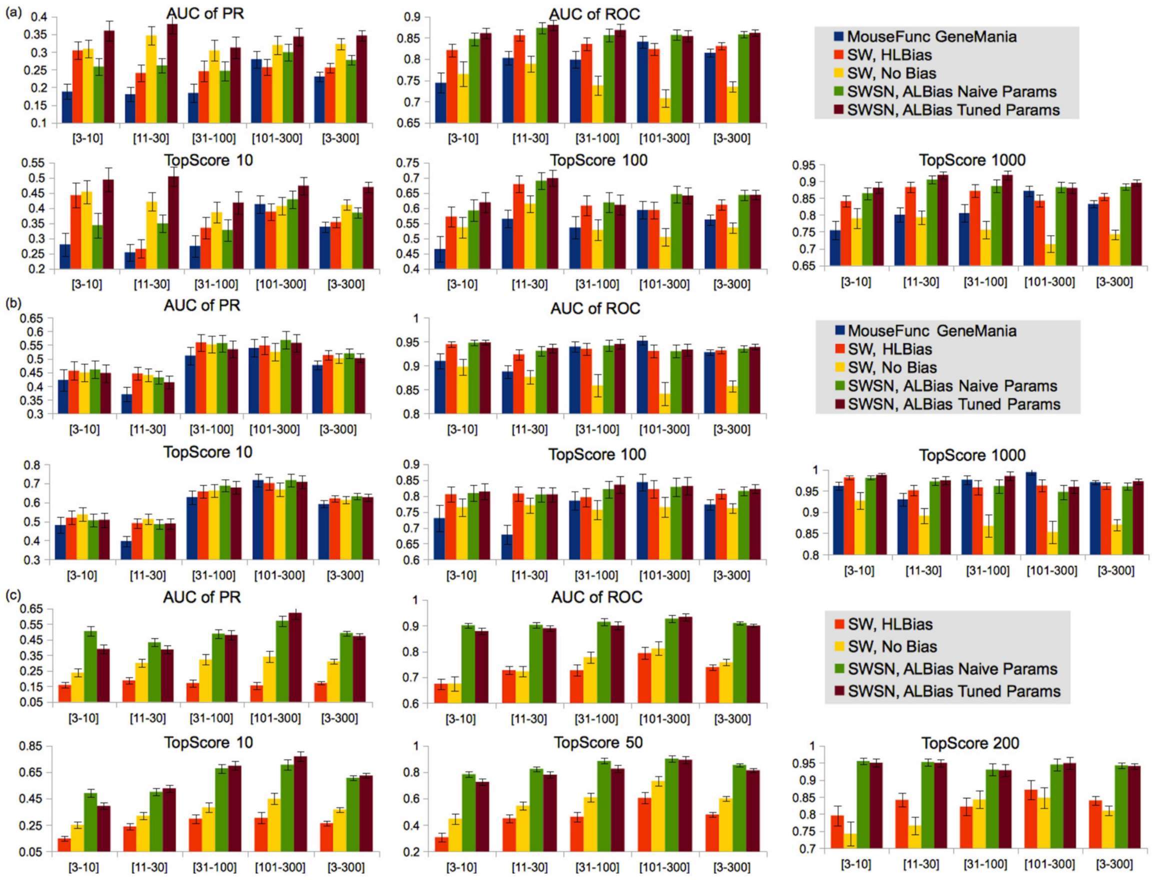

Figure S2 Performance metrics in (a) the novel scenario in mouse (488 functions, 1954 genes), (b) the test scenario in mouse (442 functions, 1718 genes), and (c) the novel scenario in yeast (511 functions, 342 genes). Metrics are presented for several buckets of specificity based on the number of observed 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.

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 GO 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.

Computational Cost

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 3.4 of the main text). 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($d*n^2+d^2*n+d^3$). 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. In such cases the complexity of SBCG reduces to O($d*n^2$) 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 in order to measure algorithmic performance. Table 4 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 categories, as well as the final norm of the residual matrix: norm(Ax-b).

Table 3. Flops and error (norm of the residual) results for the SCBG algorithm, and Conjugate Gradient (CG) algorithm applied sequentially

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.

REFERENCES

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

Mostafavi, S., and Morris, Q. (2009) Using the Gene Ontology hierarchy when predicting gene function. *UAI Conference Proceedings*, **2009**.

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

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

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.