Hierarchical classification of Gene Ontology terms using the GOstruct method

Artem Sokolov and Asa Ben-Hur
Department of Computer Science, Colorado State University
Fort Collins, CO, 80523, USA

Abstract

Protein function prediction is an active area of research in bioinformatics. And yet, transfer of annotation on the basis of sequence or structural similarity remains widely used as an annotation method. Most of today's machine learning approaches reduce the problem to a collection of binary classification problems: whether a protein performs a particular function, sometimes with a post-processing step to combine the binary outputs. We propose a method that directly predicts a full functional annotation of a protein by modeling the structure of the Gene Ontology hierarchy in the framework of kernel methods for structured-output spaces. Our empirical results show improved performance over a BLAST nearest-neighbor method, and over algorithms that employ a collection of binary classifiers as measured on the Mousefunc benchmark dataset.

1 Introduction

Protein function prediction is an active area of research in bioinformatics [25]; and yet, transfer of annotation on the basis of sequence or structural similarity remains the standard way of assigning function to proteins in newly sequenced organisms [20]. The Gene Ontology (GO), which is the current standard for annotating gene products and proteins, provides a large set of terms arranged in a hierarchical fashion that specify a gene-product's molecular function, the biological process it is involved in, and its localization to a cellular component [12]. GO term prediction is therefore a hierarchical classification problem, made more challenging by the thousands of annotation terms available. Computational methods for annotating protein function have predominantly followed the "transfer-of-annotation" paradigm where GO keywords are transferred from one protein to another based on sequence similarity [20]. This is generally done by employing a sequence alignment tool such as BLAST [1] to find annotated proteins that have a high level of sequence similarity to an un-annotated query protein. There has also been an effort to recognize "good" BLAST hits, from which annotations can then be transferred [38]. Transfer of annotation methods have several shortcomings: transfer of multiple GO keywords between proteins is not always appropriate, e.g. in the case of multi-domain proteins [11], and they fail to exploit the underlying hierarchical structure of the annotation space. Furthermore, from a machine learning perspective, transfer of annotation is a form of nearest-neighbor classifier, which is not the state-of-the-art in performance.

Prediction of protein function has been approached as a binary classification problem using a wide array of methods that predict whether a query protein has a certain function [10, 18, 36, 24, 22]. These methods leave it to the user to combine the output of classifiers trained to recognize the different possible functions and decide which of the annotations to accept. To automate the task, several methods have been proposed for reconciling predictions from collections of binary classifiers [4, 13, 19, 23]. All these approaches require training hundreds or even thousands of classifiers, depending on the level of specificity of the predicted annotations. The most recent work in the area is an extension of the GeneMANIA method that models the problem as a set of sparsely coupled networks, one for each term in the hierarchy [21].

The Hughes Lab from the University of Toronto recently hosted a competition aimed at the prediction of protein function in the *M. musculus* species [25]. The task was to infer functional annotations using several sources of data: gene expression, protein-protein interactions, protein domain composition, phenotype data and phylogenetic profiles. The competition provided a common framework for empirical comparison of the methods employed by the participants, and the test data and the predictions submitted by the contestants have been consequently released as part of the Mousefunc benchmark dataset [25]. Nearly all of the participants employed a collection of binary classifiers, each trained to predict a particular GO category. Some of the participants further post-processed the results of the binary classification using approaches which included logistic regression [23, 19] and Bayesian networks [13]. We also note the work by Chen, *et al.*, where the algorithm treated functional annotations in their entirety by flattening their hierarchical representation to an indexing system, encoding the relationship between proteins as a graph, and then using the Boltzmann machine and simulated annealing to infer new annotations [8].

We take a different approach: rather than use a collection of binary classifiers, one for each GO term, our method is a single classifier that directly incorporates the structure of the Gene Ontology and a notion of a good hierarchical classification into the training procedure. We choose to model the problem of hierarchical classification using kernel methods for structured-output spaces, a novel methodology in machine learning that is appropriate for modeling classification tasks where the output belongs to some discrete structure [2, 33, 35]. Structured output methods represent a learning problem using a joint representation of the input-output space, and attempt to learn a *compatibility function* $f(\mathbf{x}, \mathbf{y})$ that quantifies how related is an input $\mathbf{x}$ (protein in our case) with an element $\mathbf{y}$ of the output space (set of GO terms). Several classification methods have been generalized to this framework, including Support Vector Machines (SVMs) [35]. Whereas standard kernel methods use a mapping of inputs to a feature space that is represented by a kernel function [6], in the structured output setting the kernel becomes a joint function of both inputs and outputs [35]. Structured-output methods have been applied to a variety of problems, including applications in natural language processing [35, 29], and prediction of disulfide connectivity [32]. Structured-output methods have been recently applied to prediction of enzyme function [3]. Enzyme function is described by a four-level hierarchy that includes a few hundred classes, in comparison to GO which is a deep hierarchy with thousands of terms. The empirical results in the literature demonstrate that incorporating the structure of the output space into learning often leads to better performance over local learning via independent binary classifiers [27, 7].

In this work we report results using several flavors of kernel methods for structured output spaces, in a methodology we call *GOstruct*. More specifically, we focus on the structured-perceptron [9] and the structured SVM [35]. Our results demonstrate that learning the structure of the output space yields improved performance over transfer of annotation when both are given the same input-space information (BLAST hits). Additionally, we obtain a large improvement in performance on the Mousefunc dataset [25] when compared to published results.

2 Methods

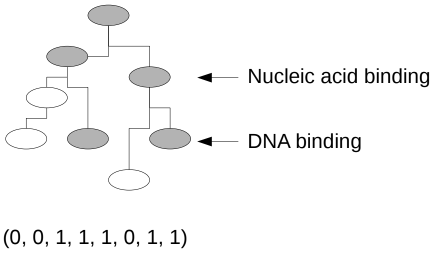

Figure 1: A schematic representation of the hierarchical label space for GO term annotation. Given that a protein is associated with a particular node in the GO hierarchy (e.g. DNA binding), it is also associated with all its ancestors in the hierarchy (including the direct parent of *DNA binding* which is *Nucleic acid binding*). The collection of GO terms associated with a protein (shaded in the figure) correspond to the nonzero entries in the vector representing the annotations.

The Gene Ontology provides annotation terms that are arranged hierarchically in three namespaces that describe different aspects of a protein's function: molecular function, biological process, and cellular component. Terms that appear deeper in the hierarchy provide more detailed information (see Figure 1 for illustration). Note that associating a protein with a particular term automatically implies the association with all the terms that generalize it, i.e. all its ancestors in the hierarchy. In the example in Figure 1, a DNA binder is also a nucleic acid binder.

We formulate prediction of GO terms as follows. Each protein is associated with a macro-label $\mathbf{y} = (y_1, y_2, ..., y_k) \in \{0,1\}^k$, where each micro-label y_i corresponds to one of the k nodes in one of the three GO namespaces. The micro-labels take on the value of 1 when the protein performs the function defined by the corresponding node, and 0 otherwise. We refer to such nodes as *positive*. Whenever a protein is associated with a particular micro-label, we also associate it with all its ancestors in the hierarchy, i.e. given a specific term, we associate with it all terms that generalize it. This enforces the constraint that parents of positive nodes are also positive. Throughout this paper we will refer to macro-labels as *labels* or *outputs*.

2.1 Measuring performance

Classifier performance is often measured using the error rate which reports the fraction of examples classified incorrectly. In the case of binary classification this can be expressed as an average of the indicator function $\Delta_{0/1}(y, \hat{y})$ that returns a value of 1 if the labels y and $\hat{y}$ do not match and a value of 0 otherwise. $\Delta_{0/1}(y, \hat{y})$ is known as the 0-1 loss. In the context of hierarchical classification, the 0-1 loss is not appropriate as it makes no distinction between slight and gross misclassifications. For instance, a label where the protein function is mis-annotated with its parent or sibling is a better prediction than an annotation in an entirely different part of the hierarchy. Yet, both will be treated the same way by the 0-1 loss.

A number of loss functions that incorporate taxonomical information have been proposed in the context of hierarchical classification [14, 29, 17]. These either measure the distance between labels by finding their least common ancestor in the taxonomy tree [14] or penalize the first inconsistency between the labels in a top-down traversal of the taxonomy [29]. Kiritchenko *et al.* proposed a loss function that is related to the F_1 measure which is used in information retrieval [37] and was used by Tsochantaridis *et al.* in the context of parse tree inference [35]. In what follows we present the F_1 loss function and show how it can be expressed in terms of kernel functions, thereby generalizing it to arbitrary output spaces. The F_1 measure is a combination of precision and recall, which for binary classification problems are defined as

$$F_1 = \frac{2 \cdot P \cdot R}{P + R}, \quad P = \frac{tp}{tp + fn}, \quad R = \frac{tp}{tp + fp},$$

where *tp* is the number of true positives, *fn* is the number of false negatives and *fp* is the number of false positives. Rather than expressing precision and recall over the whole set of examples, we express it relative to a single example (known as micro-averaging in information retrieval), computing the precision and recall with respect to the set of micro-labels. Given a vector of true labels ($\mathbf{y}$) and predicted labels ($\hat{\mathbf{y}}$) the number of true positives is the number of micro-labels common to both labels which is given by $\mathbf{y}^T\hat{\mathbf{y}}$. It is easy to verify that

$$P(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}^T\hat{\mathbf{y}}}{\hat{\mathbf{y}}^T\hat{\mathbf{y}}}, \quad R(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\mathbf{y}^T\hat{\mathbf{y}}}{\mathbf{y}^T\mathbf{y}}. \tag{1}$$

We can now express $F_1(\mathbf{y}, \hat{\mathbf{y}})$ as

$$F_1(\mathbf{y}, \hat{\mathbf{y}}) = \frac{2\,\mathbf{y}^T\hat{\mathbf{y}}}{\mathbf{y}^T\mathbf{y} + \hat{\mathbf{y}}^T\hat{\mathbf{y}}},$$

and define the F_1-loss as $\Delta_{F_1}(\mathbf{y}, \hat{\mathbf{y}}) = (1 - F_1(\mathbf{y}, \hat{\mathbf{y}}))$ [35]. This loss can be generalized to arbitrary output spaces by replacing dot products with kernels:

$$P(\mathbf{y}, \hat{\mathbf{y}}) = \frac{K(\mathbf{y}, \hat{\mathbf{y}})}{K(\hat{\mathbf{y}}, \hat{\mathbf{y}})} \qquad R(\mathbf{y}, \hat{\mathbf{y}}) = \frac{K(\mathbf{y}, \hat{\mathbf{y}})}{K(\mathbf{y}, \mathbf{y})}.$$

Substituting these expressions for precision and recall leads to the following generalization of the F_1-loss, which we call the *kernel loss*:

$$\Delta_{ker}(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{2K(\mathbf{y}, \hat{\mathbf{y}})}{K(\mathbf{y}, \mathbf{y}) + K(\hat{\mathbf{y}}, \hat{\mathbf{y}})}, \tag{2}$$

which reduces to the F_1-loss when using a linear kernel.

2.2 GOstruct: GO term prediction as a structured outputs problem

Before presenting the *GOstruct* method we provide a brief overview of binary and structured classification using kernel methods. A standard approach in training predictors for binary classification problems is to learn a discriminant function $f(\mathbf{x})$ and classify the input $\mathbf{x}$ according to the sign of $f(\mathbf{x})$. Since linear methods usually have efficient training algorithms, it is common to assume that the discriminant function is linear. A way to obtain a non-linear classifier, using an algorithm designed for linear discrimination is to assume that the data is mapped non-linearly into some feature-space using a function $\phi(\mathbf{x})$. A linear discriminant in this feature space will have the form $f(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x})$, where $\mathbf{w}$ is a vector of parameters. Whenever $\mathbf{w}$ can be expressed as a weighted sum over the images of the input examples, i.e. $\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}_i)$ the discriminant function becomes

$f(\mathbf{x}) = \sum_i \alpha_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x})$, which can expressed using the kernel function as $\sum \alpha_i K(\mathbf{x}_i, \mathbf{x})$. See a tutorial by Ben-Hur, *et. al* [6] for more details and pointers on kernel methods.

A binary classifier can predict *whether* a protein performs a certain function. For predicting *what* function the protein performs, i.e. the full macro-label $(y_1, y_2, ..., y_k)$, we turn to structured output learning. In this setting the discriminant function becomes a function $f(\mathbf{x}, \mathbf{y})$ of both inputs and labels, and can be thought of as measuring the compatibility of the input $\mathbf{x}$ with the output $\mathbf{y}$. We denote by $\mathcal{X}$ the space used to represent our inputs (proteins) and by $\mathcal{Y}$ the set of labels we are willing to consider, which is a subset of $\{0, 1\}^k$ for hierarchical multi-label classification. Given an input $\mathbf{x}$ in the input feature space $\mathcal{X}$, structured-output methods infer a label according to:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}} f(\mathbf{x}, \mathbf{y} | \mathbf{w}), \tag{3}$$

where the function $f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ is parametrized by the vector $\mathbf{w}$. This classification rule chooses the label $\mathbf{y}$ that is most compatible with an input $\mathbf{x}$. We assume the function is linear in $\mathbf{w}$, i.e. $f(\mathbf{x}, \mathbf{y} | \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y})$ in some space defined by the mapping ϕ. Whereas in two-class classification problems the mapping ϕ depends only on the input, in the structured-output setting it is a joint function of inputs and outputs.

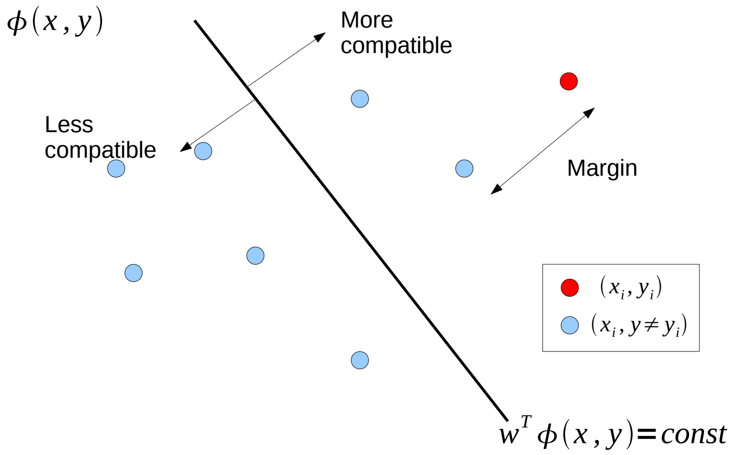

Figure 2: The geometric view of structured-output classification. We consider a given input $\mathbf{x}_i$, and plot it in the joint space of inputs and outputs in combination with different labels $\mathbf{y}$. This figure represents the ideal case: The correct label, $\mathbf{y}_i$, has the highest compatibility value with $\mathbf{x}_i$ and the second best candidate is separated by a margin. Our classifier defines a linear discriminant function over the joint input-output space defined by $\phi(\mathbf{x}, \mathbf{y})$ illustrated by the line $\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) = \text{const}$ in the figure.

To make use of kernels, we assume that the weight vector $\mathbf{w}$ can be expressed as a linear combination of the training examples:

$$\mathbf{w} = \sum_{j=1}^{n} \sum_{\mathbf{y}' \in \mathcal{Y}} \alpha_{j,\mathbf{y}'} \phi(\mathbf{x}_j, \mathbf{y}'),$$

where $\alpha_{j,\mathbf{y}'}$ is a vector of parameters indexed by j (possible input examples) and labels $\mathbf{y}'$. This leads to reparametrization of the compatibility function in terms of the coefficients α:

$$f(\mathbf{x}, \mathbf{y}|\alpha) = \sum_{j=1}^{n} \sum_{\mathbf{y}' \in \mathcal{Y}} \alpha_{j,\mathbf{y}'} K((\mathbf{x}_j, \mathbf{y}'), (\mathbf{x}, \mathbf{y})),$$

where $K : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$ is the joint kernel defined over the input-output space. For the prediction of GO terms we use a joint kernel which is a product of the input space and the output space kernels:

$$K((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = K_{\mathcal{X}}(\mathbf{x}, \mathbf{x}') K_{\mathcal{Y}}(\mathbf{y}, \mathbf{y}'). \tag{4}$$

Our intuition for using a product kernel is that two examples are similar in the input-output feature space if they are similar in both their input and the output space representations. In preliminary experiments, we also considered a second-degree polynomial kernel of the form $K((\mathbf{x}, \mathbf{y}), (\mathbf{x}', \mathbf{y}')) = (K_{\mathcal{X}}(\mathbf{x}, \mathbf{x}') + K_{\mathcal{Y}}(\mathbf{y}, \mathbf{y}'))^2$, which provided lower accuracy. For the output-space kernel, $K_{\mathcal{Y}}$, we use a linear kernel in all of our experiments; the input-space kernel is described separately for each experiment.

2.2.1 Inference

The $\arg\max$ in Equation (3) must be computed over the space of all possible labels $\mathcal{Y}$. In the context of protein function prediction, this consists of all possible combinations of a few thousand GO terms. The size of the output space is thus exponential in the size of the GO hierarchy. Explicitly enumerating all such combinations is not practical. Fortunately, a protein has only a limited number of functions. Incorporating such a limit reduces the number to be polynomial in the number of GO terms. In this paper, we further reduce this number in several ways.

During training we limited the space of labels to only those labels that appear in the training dataset. We call this space $\mathcal{Y}_1$ and argue that it makes sense to focus on learning only combinations of GO terms that occur in the training data (GO terms that tend to co-occur). This subspace contains no combinations of GO terms that are not present in the training data.

In one of our experiments we compare our approach with a BLAST nearest neighbor approach to verify that it performs at least as well as this baseline classifier. In this case, BLAST hits are the only features available to the classifier, so it is reasonable to restrict predicted labels to those that are suggested by significant BLAST hits. We define $\mathcal{Y}_3(\mathbf{x})$ to be the set of macro-labels that appear in the significant BLAST hits of protein $\mathbf{x}$ (e-values below 10^{-6}). The output space $\mathcal{Y}_2(\mathbf{x})$ is obtained by taking all the deepest nodes represented in $\mathcal{Y}_3(\mathbf{x})$ and considering the macro-labels consisting of combinations of three such nodes at the most (the average number of annotations per protein in the molecular function namespace). Prediction using $\mathcal{Y}_3(\mathbf{x})$ amounts to using only the annotations from $\mathbf{x}$'s hits, while $\mathcal{Y}_2(\mathbf{x})$ considers all combinations of annotations from $\mathbf{x}$'s hits. For example, if a protein has two significant BLAST hits, where one is labeled with GO_1 and GO_2 and the other is labeled with GO_3, then $\mathcal{Y}_2(\mathbf{x})$ consists of all nonempty subsets of $\{GO_1, GO_2, GO_3\}$, whereas $\mathcal{Y}_3(\mathbf{x})$ is $\{\{GO_1, GO_2\}, \{GO_3\}\}$. These label spaces satisfy: $\mathcal{Y}_3(\mathbf{x}) \subseteq \mathcal{Y}_2(\mathbf{x}) \subseteq \mathcal{Y}_1$.

2.3 GOstruct using the perceptron algorithm

The perceptron algorithm is one of the simplest classification methods, and its extension to the structured-outputs setting maintains this simplicity [9]. We introduce a variant of the algorithm that incorporates the concept of a margin, and propose to incorporate the loss function during

Algorithm 1 Structured Outputs Perceptron: *GOstruct*$_{p\Delta}$

```
**Input:** training data $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$
**Output:** parameters $\alpha_{i,\mathbf{y}}$ for $i = 1, ..., n$ and $\mathbf{y} \in \mathcal{Y}$.
**Initialize:** $\alpha_{i,\mathbf{y}} = 0 \quad \forall i, \mathbf{y}$. //only non-zero values of $\alpha$ are stored explicitly
**repeat**
  **for** $i = 1$ **to** $n$ **do**
    //Compute the top scoring label that differs from $\mathbf{y}_i$:
    $\bar{\mathbf{y}} \leftarrow \arg\max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} f(\mathbf{x}_i, \mathbf{y}|\alpha)$
    //Compute the margin:
    $\delta \leftarrow f(\mathbf{x}_i, \mathbf{y}_i) - f(\mathbf{x}_i, \bar{\mathbf{y}})$
    **if** $\delta < \gamma$ **then**
      $\alpha_{i,\mathbf{y}_i} \leftarrow \alpha_{i,\mathbf{y}_i} + 1$
      $\alpha_{i,\bar{\mathbf{y}}} \leftarrow \alpha_{i,\bar{\mathbf{y}}} - \Delta_{ker}(\mathbf{y}_i, \bar{\mathbf{y}})$
    **end if**
  **end for**
**until** a termination criterion is met
```

its training. Given a set of n training examples $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$, the margin-based perceptron algorithm attempts to find a vector $\mathbf{w}$ such that for each input example the true label has the largest compatibility value and the best runner-up label is separated by a user-defined margin γ:

$$\mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}_i) - \max_{\mathbf{y} \in \mathcal{Y} \setminus \mathbf{y}_i} \mathbf{w}^T \phi(\mathbf{x}_i, \mathbf{y}) \geq \gamma \quad \forall i. \tag{5}$$

The geometric intuition is shown in Figure 2.

We propose a variant of the perceptron method that incorporates the loss function in the process of training and present it as Algorithm 1. The *GOstruct* method that uses it is referred to as *GOstruct*$_{p\Delta}$, whereas the *GOstruct* method that uses the standard perceptron update is called *GOstruct*$_p$. In the standard version of the perceptron method whenever an example is misclassified or its margin is not sufficiently large, the element of the parameter vector α corresponding to the misclassification is decremented by 1 regardless of whether the classifier made a big mistake or a slight one [9, 27]. Intuitively, we would like to penalize gross misclassifications with larger values than slight errors. We propose to update the α coefficients using the amount of dissimilarity between the true and predicted labels. This can be done by utilizing $\Delta_{ker}(\mathbf{y}_i, \bar{\mathbf{y}})$, the loss between the true label $\mathbf{y}_i$ and the highest scoring candidate $\bar{\mathbf{y}}$ that differs from it. Note that the loss value is between 0 and 1. Thus, when there is no similarity between the predicted and the true labels, the corresponding α coefficient will be updated by -1, as in the traditional rule. Less penalty will be assigned for predicting labels that are more similar to the true label. In our application, the termination criterion is taken to be a limit on the number of iterations.

2.4 The Structured Support Vector Machine

The perceptron algorithm attempts to separate the true labels from the second best candidates by a fixed user-defined margin. Intuitively, the larger the margin, the more robust the decision boundary is to noise. The structured support vector machine attempts to maximize the margin, while enforcing the constraints of Equation (5). This can be alternatively formulated as minimizing the norm of the weight vector $\mathbf{w}$, while keeping the margin fixed [35]:

$$\begin{aligned}&\min \frac{1}{2}\mathbf{w}^T\mathbf{w}\\ &\text{s.t.}\quad \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}_i) - \max_{\mathbf{y}\in\mathcal{Y}\backslash\mathbf{y}_i} \mathbf{w}^T\phi(\mathbf{x}_i,\mathbf{y}) \geq 1 \quad \forall i.\end{aligned}$$

Unfortunately, this generally results in over-constrained problems with no solutions. To get around this, we employ the *n-slack formulation* of the problem [35], where we allow for some amount of margin violation. The amount of violation is represented by the slack variables ξ_i, which we add to the minimization criterion:

$$\begin{aligned}&\min_{\mathbf{w},\xi\geq 0} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \frac{C}{n}\sum_{i=1}^{n}\xi_i \qquad (6)\\ &\text{s.t.}\ \ \forall i,\ \mathbf{w}^T\delta\psi_i(\bar{\mathbf{y}}_i) \geq \Delta(\mathbf{y}_i,\bar{\mathbf{y}}_i) - \xi_i,\end{aligned}$$

where $\delta\psi_i(\mathbf{y}) = \psi(\mathbf{x}_i,\mathbf{y}_i) - \psi(\mathbf{x}_i,\mathbf{y})$ and $\bar{\mathbf{y}}_i$ is the highest scoring candidate that differs from $\mathbf{y}_i$ as before. As is the case for any other SVM, C controls the trade-off between the smoothness of the predictor and the amount of margin violation. In this formulation, called margin rescaling, the slack variables are offset by the loss function, effectively relaxing the constraints for closely related outputs. The corresponding *GOstruct* method is referred to as *GOstruct*$_{svm}$. In our experiments we have also considered the slack rescaling formulation [35], which achieved results comparable to those that employed margin rescaling, however took longer to compute.

To train a structured SVMs, we used the working set approach [35] with an SMO-like algorithm [26] as the underlying optimizer. Like the perceptron algorithm, all structured SVM formulations were expressed in terms of the dual coefficients α. We refer the reader to the original papers [35, 15] for details.

3 Data Preparation and Experimental Setup

We performed two experiments: one comparing the *GOstruct* methods to homology-based transfer-of-annotation, and another comparing its performance on the Mousefunc dataset.

3.1 Four species prediction-by-sequence-similarity experiment

To compare the *GOstruct* algorithms to the transfer-of-annotation method, we computed sequence similarity using BLAST for the following four species: *C. elegans*, *D. melanogaster*, *S. cerevisiae*, and *S. pombe*. Sequence data was obtained from the genome database of each organism (`http://www.wormbase.org/`,
`http://flybase.bio.indiana.edu/`, `http://www.yeastgenome.org/`) and annotations were obtained from the Gene Ontology website at `http://www.geneontology.org`. Our experiments follow the leave-one-species-out paradigm [38], where we withhold one species for testing and train the *GOstruct* method on the remaining data, rotating the withheld species. This variant of cross-validation simulates the situation of annotating a newly-sequenced genome.

To prepare the data we removed all annotations that were discovered through computational means as these are generally inferred from sequence or structure similarity and would introduce bias into any classifier that used sequence similarity to make a prediction [28]. This was done

Test on	*C. elegans*	*D. melanogaster*	*S. cerevisiae*	*S. pombe*	Output
# proteins	926	1893	1907	939	Space
BLAST-NN	0.64	0.47	0.39	0.37	
$GOstruct_p$	0.61	0.43	0.40	0.41	$\mathcal{Y}_1$
$GOstruct_p$	0.61	0.43	0.38	0.37	$\mathcal{Y}_2$
$GOstruct_p$	0.61	0.42	0.37	0.37	$\mathcal{Y}_3$
$GOstruct_{p\Delta}$	0.58	0.38	0.38	0.37	$\mathcal{Y}_1$
$GOstruct_{p\Delta}$	0.58	0.37	0.36	0.34	$\mathcal{Y}_2$
$GOstruct_{p\Delta}$	0.60	0.41	0.37	0.34	$\mathcal{Y}_3$
$GOstruct_{svm}$	0.60	0.42	0.38	0.37	$\mathcal{Y}_1$
$GOstruct_{svm}$	0.61	0.41	0.36	0.34	$\mathcal{Y}_2$
$GOstruct_{svm}$	0.61	0.42	0.36	0.35	$\mathcal{Y}_3$
Random	0.82	0.87	0.88	0.82	

Table 1: Classification results on predicting GO molecular function terms (361 terms that have more than 10 annotations). We compare BLAST-NN with two variants of the perceptron ($GOstruct_p$ and $GOstruct_{p\Delta}$) and one SVM variant $GOstruct_{svm}$), both using the margin re-scaling formulation. across three methods for limiting the output space. Reported is mean kernel loss per protein for each algorithm. The number of proteins used in each organism is displayed in the second row. For comparison, we also include the performance of a random classifier that transfers annotation from a training example chosen uniformly at random. The standard deviations of these results are in the range 0.003-0.01; see text for details.

by removing all annotations with the evidence codes: IEA, ISS, ND, RCA, and NR. Note that considering only annotations that were derived by biological experiments limits the number of species that can be considered to a very small set of well-studied model organisms, and for simplicity we focused on the eukaryotes listed above. After filtering for evidence codes we considered all GO molecular function terms that appear as annotations in at least 10 proteins, resulting in a total of 361 nodes. Note that method development was performed using the GO-slims ontology, thereby avoiding overfitting our test data.

We then ran BLAST for each of the proteins in our dataset against all four species, removing the hits where a protein was aligned to itself. We employed the nearest-neighbor BLAST methodology as our baseline. For every test protein, we transferred the annotations from the most significant BLAST hit against a protein from another species. Proteins which didn't have a hit with an e-value below 10^{-6} were not considered in our experiments.

The *GOstruct* methods are provided exactly the same data as the BLAST method: each protein was represented by its BLAST scores against the database proteins; this is known as the BLAST empirical kernel map [30]; more specifically, features were the negative-log of the BLAST e-values below 50, and features were then normalized to have values less than 1.0. An empirical kernel map arises from the intuition that two similar proteins will have similar patterns of similarity to proteins in the database, i.e. their vectors of e-values will be similar.

We ran five fold cross-validation on the training data to select a suitable value of the margin parameters γ (for perceptron) and C (for structured SVM) for each left-out species. In our experiments, we noticed that finding the right value of γ for the perceptron algorithm was not as essential as using the loss update proposed in the previous section.

3.2 Mousefunc experiment

As a further comparison of the *GOstruct* method we ran it on the Mousefunc dataset, using exactly the same data that was provided to the participants [25]. This includes two different sources of gene expression data, protein-protein interaction adjacency matrix, protein pattern annotation data from Pfam and InterPro, and phylogenetic profiles. We normalized the gene expression data by subtracting the mean and dividing by the standard deviation of each feature. Additionally, we treated missing entries in any source of data as zero. The features within each source of data were normalized to unit vectors to normalize the contribution of each data source to the overall input space kernel, $K_{\mathcal{X}}$, which was computed as the sum of linear kernels over the individual datasets. As before, the joint kernel is computed as a product of a linear output space kernel $K_{\mathcal{Y}}$ and $K_{\mathcal{X}}$ (c.f. Equation 4). In this experiment we only considered the output space $\mathcal{Y}_1$ which considers only output labels that occur in the training data since the dataset does not include labels of proteins with sequence similarity.

We trained the *GOstruct* methods to predict annotations for the subset of GO terms requested by the competition organizers. Any training or test examples that had no annotations in this subset were removed from the analysis. Analysis of molecular function, biological process and cellular component namespaces were performed separately from each other.

The values of γ and C were again chosen by performing cross-validation on the training data. We noticed that the algorithms were quite sensitive to the choice of their parameters on this dataset.

4 Results

4.1 Four species experiment

The results for the leave-one-species-out experiments are presented in Table 1. The results show that the various flavors of the *GOstruct* method outperform the BLAST nearest-neighbor classifier (BLAST-NN), except for the standard implementation of the perceptron method ($GOstruct_p$), which performs only marginally better than BLAST-NN. Before looking at the differences between the *GOstruct* methods, we note that all the classifiers performed poorly on *C. elegans*. This is due to the fact that a vast majority of proteins in this species are annotated as protein binders (GOID:0005515). Such annotations contain little information from a biological standpoint and result in a skewed set of labels.

Our first observation is that the $GOstruct_{p\Delta}$ method, which uses the loss function in the update rule of the perceptron, outperformed $GOstruct_p$. Furthermore, excluding *C. elegans*, restricting inference to the sets $\mathcal{Y}_2$ or $\mathcal{Y}_3$ resulted in better performance than using the set $\mathcal{Y}_1$ for all the flavors of the *GOstruct* method. The larger label-space, $\mathcal{Y}_1$, results in the inference procedure considering many annotations that are irrelevant to the actual function of the protein, which can reduce prediction accuracy. When used in conjunction with $\mathcal{Y}_2$ or $\mathcal{Y}_3$ our structured-outputs methods can be thought of as prioritizing the annotations suggested by BLAST in a way that uses the structure of the Gene Ontology hierarchy. Although inference using $\mathcal{Y}_2(\mathbf{x})$ and $\mathcal{Y}_3(\mathbf{x})$ gave very similar results, $\mathcal{Y}_2(\mathbf{x})$, which provides a richer set of options, yielded better performance than $\mathcal{Y}_3(\mathbf{x})$ in 7 cases, while $\mathcal{Y}_3(\mathbf{x})$ provided a better result in only one case.

The SVM-based algorithm outperforms BLAST-NN and $GOstruct_p$, but not $GOstruct_{p\Delta}$, which is a significantly simpler algorithm. While only the margin re-scaling results are reported, similar performance was achieved with slack re-scaling.

GO namespace	number of terms	test examples	annotations	
			train	test
MF	205	531	3.2	3.3
BP	513	626	7.1	8.0
CC	119	307	3.4	3.7

Table 2: Statistics of the Mousefunc dataset across namespaces: molecular function (MF), biological process (BP), and cellular component (CC). We provide the number of terms in each namespace for which annotations were provided, the number of examples in the test set and the average number of annotations per protein in the training and test sets.

We assessed the robustness and variability of the results by randomly sampling the data for training and testing: 20% of the training data was chosen at random and withheld from training. The classifier was then trained on the remaining 80% of the training data and tested as before. This provided us with a standard deviation measure that indicated how consistent the classifiers were at obtaining the performance presented in Table 1. We computed the standard deviations across 30 trials for every classifier. The values for the different classifiers were between 0.003 and 0.01. The differences in performance between BLAST-NN and the *GOstruct* methods (except for the naive perceptron method) are all greater than the observed variability.

In summary, the results in Table 1 support our hypothesis that learning the structure of the output space is superior to performing transfer of annotation. The *GOstruct* methods have the added advantage that other sources of relevant genomic data can be modeled in this framework as shown in the next set of experiments.

4.2 Mousefunc experiment

We applied the SVM and perceptron-based *GOstruct* methods to the Mousefunc challenge dataset whose statistics are provided in Table 2. The *GOstruct* method produces a set of annotations for each protein, from which we directly computed the average kernel loss per example and precision/recall averaged across GO terms. The results are provided in Table 3. The predictions made by competitors in the Mousefunc challenge consist of confidence scores for each GO term across all proteins. Computing precision and recall therefore requires thresholding the predictions at some level. We chose to set the threshold such that the number of predictions for each term equals the number of proteins annotated with that term in the test set. In this case, precision is equal to recall; for the *GOstruct* method precision and recall are very close, thereby making its results comparable to those of the Mousefunc competitors. Note that this gives a significant advantage to the methods we are comparing to, as they have access to information about the test set. But despite this advantage, the *GOstruct* method outperforms all the other methods except Funckenstein by a large margin in all namespaces (Table 3). For algorithms that produce confidence scores the kernel loss can be computed directly from the confidence scores, without thresholding the predictions. For most algorithms (except GeneMania), thresholding the confidence measures leads to higher kernel loss.

To further illustrate the contribution of using the structured SVM approach, we compare the *GOstruct* method to a collection of independent binary SVMs. The SVM-based $GOstruct_{svm}$ method outperforms the collection of binary SVMs under all the performance measures (Table 2). The binary SVM experiment was performed using the SVM implementation in the `PyML` machine

GO	Perf.	Literature							*GOstruct*		SVM
nmspc.	measure	Alg 1	Alg 2	Alg 3	Alg 4	Alg 5	Alg 6	Alg 7	$p\Delta$	*svm*	(binary)
MF	kernel loss^r	0.63	0.47	0.43	0.67	0.52	0.62	**0.33**	0.52	**0.33**	0.42
	precision	0.45	0.61	0.53	0.56	0.51	0.57	0.66	0.40	0.67	0.59
	recall	0.45	0.61	0.53	0.56	0.51	0.57	0.66	0.52	0.65	0.59
	kernel loss	0.61	0.42	0.50	0.46	0.52	0.47	0.39			0.42
BP	kernel loss^r	0.79	0.69	0.63	0.84	0.74	0.72	**0.58**	0.68	0.60	0.67
	precision	0.15	0.25	0.23	0.27	0.20	0.29	0.31	0.15	0.28	0.27
	recall	0.15	0.25	0.23	0.27	0.20	0.29	0.31	0.24	0.28	0.27
	kernel loss	0.86	0.68	0.71	0.71	0.76	0.67	0.65			0.67
CC	kernel loss^r	0.60	0.60	0.53	0.76	0.67	0.66	**0.50**	0.62	**0.50**	0.59
	precision	0.33	0.36	0.39	0.42	0.35	0.40	0.46	0.23	0.43	0.45
	recall	0.33	0.36	0.39	0.42	0.35	0.40	0.46	0.42	0.46	0.45
	kernel loss	0.76	0.62	0.63	0.60	0.68	0.62	0.59			0.59

Table 3: Prediction results on the Mousefunc dataset for molecular function (MF), biological process (BP) and cellular component (CC) namespaces. Reported are the the mean kernel loss per protein and precision/recall Lower values of the loss and higher values of precision/recall are better. The best value for each experiment is highlighted. There are two lines with kernel loss results. The results labeled as kernel loss^r are obtained using the raw confidence scores with no thresholding. All the other results in the table are obtained by thresholding competitor results. *GOstruct* predictions require no thresholding, so only one set of kernel loss numbers is reported. Alg 1 denotes the work by Kim, *et al.* [16]. Alg 2 is an ensemble of calibrated SVMs by Obozinski, *et. al* [23]. Alg 3 is the kernel logistic regression, submitted by Lee, *et al.* [19]. Alg 4 is geneMANIA [22]. Alg 5 is GeneFAS [8]. Alg 6 is the work by Guan, *et al.* [13]. Alg 7 is Funckenstein [34]. $GOstruct_{p\Delta}$ uses the perceptron algorithm (Algorithm 1), and $GOstruct_{svm}$ denotes the n-slack formulation of the structured SVMs with margin re-scaling. The last column presents the results of running binary SVMs on each node individually. The variability in our results was computed as in the previous experiment and yielded a standard deviation of 0.008 for the perceptron, and 0.02 for the SVMs.

learning library available at `pyml.sf.net` run with the default parameters, and the same input-space kernel used to assess the *GOstruct* methods.

In a preliminary version of this paper we reported ROC scores (area under the ROC curve) in addition to the kernel loss [31]. There is no accepted definition of ROC curves for structured output methods, which here requires the definition of a confidence measure for a component of the overall prediction—confidence in the prediction of a particular term (micro-label).

A full run of the $GOstruct_{svm}$ method (including model selection) took 6 hours for the molecular function namespace, 30 hours for biological process, and 1.5 hours for cellular component (all numbers are user time, and experiments were performed on a 3.0GHz 8Gb RAM workstation using a single core). The perceptron-based method took about half the time. The computation time across namespaces is affected by the number of annotation terms: The arg max operation in Eqn 3 requires a traversal over all combinations of terms seen in the training set. Proteins are also annotated with more terms in the biological process namespace (see Table 2) which increases the computation time of the output-space kernel.

The perceptron-based $GOstruct_{p\Delta}$ algorithm is significantly simpler than the majority of the algorithms employed by the participants in the Mousefunc challenge. It is very easy to describe and implement, and converged quickly (usually in as few as five passes through the training data). While being significantly simpler than all other algorithms, it maintained competitive performance with the other entries. Whereas in the four species experiment the structured perceptron was slightly better than the structured-SVM, the structured-SVM was much better on the Mousefunc data. We believe this has to do with the sparsity of the data in the four-species experiment: each protein has appreciable levels of similarity to only a handful of other proteins; the Mousefunc data on the other hand is not sparse. In our experience, simple algorithms (e.g. perceptron or nearest-neighbor) often perform very well when data is sparse.

All of the algorithms we looked at in this work performed best when tasked with the prediction

GO namespace	Inference method	$GOstruct_{p\Delta}$ Loss	Prc.	Rec.	$GOstruct_{svm}$ Loss	Prc.	Rec.
MF	independent	0.52	0.40	0.52	0.33	0.67	0.65
	combined	0.50	0.42	0.52	0.40	0.66	0.61
BP	independent	0.67	0.15	0.24	0.60	0.28	0.28
	combined	0.70	0.15	0.20	0.66	0.31	0.24
CC	independent	0.62	0.23	0.42	0.50	0.43	0.46
	combined	0.67	0.35	0.37	0.65	0.51	0.37

Table 4: Prediction across GO namespaces. We compare our original results for classifying each namespace independently (the first row for each namespace in the table, labeled as "independent") with simultaneous prediction across all namespaces (the second row for each namespace in the table, labeled as "combined"). Presented are kernel loss, precision and recall values for two of the *GOstruct* classifiers.

of molecular function, followed by cellular component, with worst performance on prediction of the biological process namespace annotations. There are several factors that may contribute to this ranking: molecular function is often associated with specific sequence patterns (part of the input in the Mousefunc data), making it the easiest to predict. Biological process is the namespace that has the largest number of terms which adds to the difficulty—the classifier has more ways of making a wrong prediction. In experiments published elsewhere in predicting individual GO terms from sequence using a BLAST-NN approach, performance in the cellular component and biological process namespace was very similar, and as we observe here, accuracy was much higher in the molecular function namespace [28].

4.2.1 Performance by term specificity

As in the original Mousefunc competition paper, we investigated the dependence of classifier performance on the number of examples available per GO term [25]. We divided GO terms into four categories by the number of examples available for training: {3-10} examples, {11-30} examples, {31-100} examples, and {101-300} examples, and computed average precision/recall for each category separately. In general, terms with fewer training samples are more specific, as indicated by them being deeper in the hierarchy (see Table S.2 in the supplementary material at `http://www.cs.colostate.edu/~asa/supplements/gostruct/jbcb/supplement.txt`). When evaluating prediction accuracy on molecular function GO terms for which only three to ten training examples are available, the $GOstruct_{svm}$ method achieves 85% precision, with the next runner-up (Alg 2) at 64%. A similar pattern can be observed in the biological process namespace. As the number of available training samples grows, $GOstruct_{svm}$ maintains its competitive edge, being occasionally outperformed by the other algorithms, which were again thresholded on the basis of the test data. Detailed results from this experiment are available in the online supplement at `http://www.cs.colostate.edu/~asa/supplements/gostruct/jbcb/supplement.txt`.

From our experience in predicting enzyme function, we expect very specific GO molecular function terms to be associated with highly specific sequence patterns, allowing highly accurate predictions [5]. But with very few training examples available, the accuracy of a binary classifier trained to predict such GO terms suffers from the lack of training data. We believe that by learning a single classifier for the whole hierarchy the *GOstruct* method is able to perform better on GO terms for which little training data is available since it can leverage information from related terms.

4.3 Prediction across GO namespaces

In the results presented thus far predictions were made independently in each namespace. However, protein annotations are correlated across namespace. For example, proteins that participate in DNA replication are likely to be localized to the cell's nucleus. We therefore performed an additional experiment where we train a classifier to predict GO terms in all three namespaces simultaneously. This directly implements the above observation, since in training we consider only combinations of GO terms that occur in the training data.

The results from this experiment are presented in Table 4. The results labeled as "independent" are taken from Table 3 and are included for comparison. These correspond to predicting the keywords from each namespace separately. The results labeled as "combined" correspond to the classifier that was trained on all namespaces simultaneously. We then measured the accuracy of each full prediction with respect to each namespace. We observe that in all cases simultaneous prediction has higher precision and lower recall, with the overall kernel loss being higher, except in one case (prediction of biological process using the perceptron). This can be understood as follows: by only considering combinations of GO terms that occur across namespaces in the training data our predictions become more accurate; but this reduced flexibility leads to lower recall since the GO term combinations present in the training data likely do not fully represent all the relevant combinations of GO terms. For future work we will put together a more comprehensive list of GO-term combinations observed in other organisms which will likely better capture GO-term co-occurrence.

We considered a second approach for making inference across namespaces which incrementally predicts annotations namespace by namespace, with predicted annotations serving as input for the next stage in the prediction process. In the first stage we train a classifier to predict annotations in a given namespace. The annotations predicted by this classifier are then used as input features for prediction of annotations in another namespace. And finally, results from two namespaces are used to infer annotations in the remaining namespace. For prediction of biological process using molecular function predictions the precision and recall were 0.29 and 0.27, respectively compared to 0.28 and 0.28 for the prediction made using an independent classifier. For prediction of cellular component using molecular function predictions the precision and recall were 0.44 and 0.47, respectively compared to 0.43 and 0.46 for the prediction made using an independent classifier. All the other results were not as accurate as using independent classifiers (see detailed results in the online supplement). We believe this is the result of accumulation of errors—the stagewise classifier is useful only when using molecular function as data, since this is the namespace where predictions are most accurate. Otherwise, the process adds too much noise.

5 Conclusions

In this paper we presented the *GOstruct* method for predicting GO terms which explicitly models the structure of the GO hierarchy using kernel methods for structured output spaces, a novel development in the field of machine learning. The *GOstruct* method outperforms both the traditional transfer-of-annotation method when provided only with sequence similarity, and outperforms by a large margin all but one of the algorithms tested in the recent Mousefunc function prediction challenge. The only method whose performance is competitive with the performance of *GOstruct* was the Funckenstein method, and only when its thresholding was computed using using knowledge of the true number of labels of each protein.

Through the use of structured output methods we could incorporate into the training procedure a loss function which allows us to directly optimize the accuracy of hierarchical classification. This has two advantages over training a collection of binary classifiers. First, we don't need to reconcile potentially conflicting predictions from multiple classifiers. Second, we are not required to come with a threshold for deciding whether a particular annotation is associated with a given protein, a non-trivial problem faced by users of binary classifiers that report a confidence measure.

A well-known issue in the structured-output approach is the need to consider a potentially exponential number of outputs during inference. We proposed several ways for limiting the size of the search space, and found that this not only leads to efficient inference and training, but also improves classifier accuracy.

In future work we plan to extend the *GOstruct* framework in several ways: integrate unlabeled data through semi-supervised learning, consider additional methods for performing inference within a GO namespace, and across namespaces, and explore ways of probing the classifier to determine features that are responsible for the way a protein was classified.

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, and D.J. Lipman. Basic local alignment search tool. *J. Mol. Biol*, 215(3):403–410, 1990.

[2] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *20th International Conference on Machine Learning (ICML)*, 2003.

[3] K. Astikainen, L. Holm, E. Pitk
"anen, S. Szedmak, and J. Rousu. Towards structured output prediction of enzyme function. In *BMC proceedings*, volume 2, page S2. BioMed Central Ltd, 2008.

[4] Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.

[5] A. Ben-Hur and D. Brutlag. Protein sequence motifs: Highly predictive features of protein function. In I. Guyon, S. Gunn, M. Nikravesh, and L. Zadeh, editors, *Feature extraction, foundations and applications*. Springer Verlag, 2006.

[6] A. Ben-Hur, C.S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support Vector Machines and Kernels for Computational Biology. *PLoS Computational Biology*, 4(10), 2008.

[7] N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental algorithms for hierarchical classification. *The Journal of Machine Learning Research*, 7:31–54, 2006.

[8] Y. Chen and D. Xu. Global protein function annotation through mining genome-scale data in yeast *Saccharomyces cerevisiae*. *Nucleic acids research*, 32(21):6414, 2004.

[9] M. Collins. Discriminative training methods for hidden Markov models: theory and experiments with perceptron algorithms. *Proceedings of the ACL-02 conference on Empirical methods in natural language processing-Volume 10*, pages 1–8, 2002.

[10] M. Deng, T. Chen, and F. Sun. An integrated probabilistic model for functional prediction of proteins. In *RECOMB*, pages 95–103, 2003.

[11] Michael Y. Galperin and Eugene V. Koonin. Sources of systematic error in functional annotation of genomes: Domain rearrangement, non-orthologous gene displacement and operon disruption. *In Silico Biology*, 1(1):55–67, 1998.

[12] Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.*, 25(1):25–9, 2000.

[13] Y. Guan, C. Myers, D. Hess, Z. Barutcuoglu, A. Caudy, and O. Troyanskaya. Predicting gene function in a hierarchical context with an ensemble of classifiers. *Genome Biology*, 9(Suppl 1):S3, 2008.

[14] T. Hofmann, L. Cai, and M. Ciaramita. Learning with taxonomies: Classifying documents and words. In *NIPS Workshop on Syntax, Semantics, and Statistics*, 2003.

[15] T. Joachims, T. Finley, and C.N.J. Yu. Cutting-plane training of structural SVMs. *Machine Learning*, 2009.

[16] W. Kim, C. Krumpelman, and E. Marcotte. Inferring mouse gene functions from genomic-scale data using a combined functional network/classification strategy. *Genome Biology*, 9(Suppl 1):S5, 2008.

[17] Svetlana Kiritchenko, Stan Matwin, and A. Fazel Famili. Functional annotation of genes using hierarchical text categorization. In *Proc. of the BioLINK SIG: Linking Literature, Information and Knowledge for Biology, a joint meeting of the ISMB BioLINK Special Interest Group on Text Data Mining and the ACL Workshop on Linking Biological Literature, Ontologies and Databases*, 2005.

[18] G. R. G. Lanckriet, T. De Bie, N. Cristianini, M. I. Jordan, and W. S. Noble. A statistical framework for genomic data fusion. *Bioinformatics*, 20(16):2626–2635, 2004.

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

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

[21] S. Mostafavi and Q. Morris. Using the gene ontology hierarchy when predicting gene function. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, Montreal, Canada, 2009.

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

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

[24] D. Pal and D. Eisenberg. Inference of protein function from protein structure. *Structure*, 13:121–130, January 2005.

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

[26] J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 208, 1999.

[27] V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and inference over constrained output. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129, 2005.

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

[29] J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-based learning of hierarchical multilabel classification models. *The Journal of Machine Learning Research*, 7:1601–1626, 2006.

[30] B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W.S. Noble. A kernel approach for learning from almost orthogonal patterns. In *Proceedings of the 13th European Conference on Machine Learning*, pages 511–528. Springer-Verlag London, UK, 2002.

[31] A. Sokolov and A. Ben-Hur. GOstruct: utilizing the structure of the gene ontology for accurate prediction of protein function. In *8th Annual International Conference on Computational System Bioinformatics (CSB)*, 2009.

[32] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediction models: A large margin approach. In *Twenty Second International Conference on Machine Learning (ICML05)*, 2005.

[33] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, volume 16, page 51. MIT Press, 2004.

[34] W. Tian, L. Zhang, M. Taşan, F. Gibbons, O. King, J. Park, Z. Wunderlich, J.M. Cherry, and F. Roth. Combining guilt-by-association and guilt-by-profiling to predict *Saccharomyces cerevisiae* gene function. *Genome Biology*, 9(Suppl 1):S7, 2008.

[35] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun. Large margin methods for structured and interdependent output variables. *The Journal of Machine Learning Research*, 6:1453–1484, 2005.

[36] K. Tsuda, H.J. Shin, and B. Schölkopf. Fast protein classification with multiple networks. In *ECCB*, 2005.

[37] CJ Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann Newton, MA, USA, 1979.

[38] A. Vinayagam, R. K onig, J. Moormann, F. Schubert, R. Eils, K.-H. Glatting, and S. Suhai. Applying support vector machines for gene ontology based gene function prediction. *BMC Bioinformatics*, 5:178, 2004.