

# GOstruct: PREDICTION OF GENE ONTOLOGY TERMS USING METHODS FOR STRUCTURED OUTPUT SPACES

Artem Sokolov and Asa Ben-Hur\*

*Department of Computer Science, Colorado State University*

*Fort Collins, CO, 80523, USA*

*Email: {sokolov, asa}@cs.colostate.edu*

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

Biologists have come to rely on annotations of protein function during their work. Since determining the function of a protein experimentally is an expensive and laborious process, a large fraction of the annotations in current databases are predicted computationally<sup>26</sup>. Transfer of annotation on the basis of sequence or structural similarity has proven to be an effective way of annotating proteins, and remains the standard way of assigning function to proteins in newly sequenced organisms<sup>18</sup>. In recent years more sophisticated machine learning methods are being applied to this problem<sup>22, 8, 3, 11, 20, 17</sup>.

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<sup>10</sup>. Until recently, 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<sup>18</sup>. This is generally done by employing a sequence alignment tool such as BLAST<sup>1</sup> 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 rec-

ognize “good” BLAST hits, from which annotations can then be transferred<sup>34</sup>. 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<sup>9</sup>, and they fail to exploit the underlying hierarchical structure of the annotation space.

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<sup>8, 16, 32, 21</sup>. 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 predictors employ Bayesian networks or logistic regression to infer the most likely set of annotations in a top-down fashion<sup>3, 11, 17, 20</sup>. All these approaches require training hundreds or even thousands of classifiers, depending on the level of specificity of the predicted annotations.

The Hughes Lab from the University of Toronto recently hosted a competition aimed at the prediction of protein function in the *M. musculus* species<sup>22</sup>. 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 com-

---

\*Corresponding author.

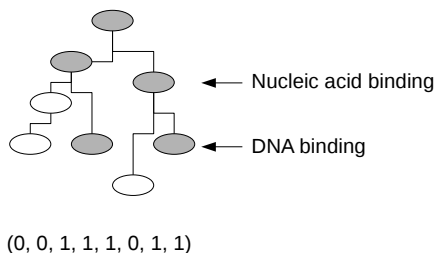
petition provided a common framework for empirical comparison of the predictors employed by the participants, and the test data and the predictions submitted by the contestants have been consequently released as part of the Mousefunc benchmark dataset<sup>22</sup>. Nearly all of the participants employed 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<sup>20, 17</sup> and Bayesian networks<sup>11</sup>. 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<sup>6</sup>.

In this paper we take a different approach: rather than use a collection of binary classifiers, we construct a single classifier that learns to predict a set of GO terms, instead of independently predicting individual ones. Since GO terms form a hierarchy, and a protein can be associated with several GO terms, GO term prediction can be formulated as a hierarchical multi-label classification problem. We choose to model this problem using structured-output classification methods, a novel methodology in machine learning that is appropriate for modeling classification tasks where the output belongs to some discrete structure<sup>2, 30, 31</sup>. 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)<sup>31</sup>. Furthermore, we choose to use *kernel methods* for structured output methods. Whereas kernel methods use a mapping of inputs to a feature space that is represented by a kernel function<sup>4</sup>, in the structured output setting the kernel becomes a joint function of both inputs and outputs<sup>31</sup>. Structured-output methods have been applied to a variety of problems, including applications in natural language processing<sup>31, 27</sup> and pre-

diction of disulfide connectivity<sup>29</sup>. 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 binary classifiers, when the binary tasks are not independent<sup>25, 5</sup>.

In this work we report results on our experiments using several flavors of kernel methods for structured output spaces, in a methodology we call *GOstruct*. More specifically, we focus on the structured-perceptron<sup>7</sup> and the structured SVM<sup>31</sup>. 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 improved performance on the Mousefunc dataset<sup>22</sup> when compared to published results.

## 2. METHODS



**Fig. 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 nodes associated with a protein (shaded in the figure) is represented by a vector of labels where nonzero entries correspond to the GO nodes associated with the protein.

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 associa-

tion 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.

In view of the above discussion, we formulate prediction of GO terms as follows. Each protein is associated with a macro-label  $\mathbf{y} = (y_1, y_2, \dots, y_k) \in \{0, 1\}^k$ , 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. 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*. Our focus in this work will be on predicting GO terms for each namespace separately.

## 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 assigned the same loss since they don't match the true label.

A number of loss functions that incorporate taxonomical information have been proposed in the context of hierarchical classification<sup>12, 27, 15</sup>. These either measure the distance between labels by finding their least common ancestor in the taxonomy tree<sup>12</sup> or penalize the first inconsistency between the labels in a top-down traversal of the taxonomy<sup>27</sup>. Kiritchenko *et al.* proposed a loss function that is related to the  $F_1$  measure which is used in information retrieval<sup>33</sup> and was used by Tsochantaridis *et*

*al.* in the context of parse tree inference<sup>31</sup>. 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 two-class 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

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

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  $\Delta_{F_1}(\mathbf{y}, \hat{\mathbf{y}}) = (1 - F_1(\mathbf{y}, \hat{\mathbf{y}}))$ <sup>31</sup>. We propose to generalize this loss to arbitrary output spaces by making use of kernels. Replacing dot products with kernels we obtain

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

If we use a linear kernel, these definitions yield values identical to those in Equation (1). Expressing precision and recall using kernels leads to the following generalization of the  $F_1$ -loss, which we call the *kernel loss*:

$$\Delta_{ker}(\mathbf{y}, \hat{\mathbf{y}}) = 1 - F_1(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{2K(\mathbf{y}, \hat{\mathbf{y}})}{K(\mathbf{y}, \mathbf{y}) + K(\hat{\mathbf{y}}, \hat{\mathbf{y}})}. \quad (2)$$

We used the kernel loss to measure accuracy in our experiments.

## 2.2. GOstruct: GO term prediction as a structured outputs problem

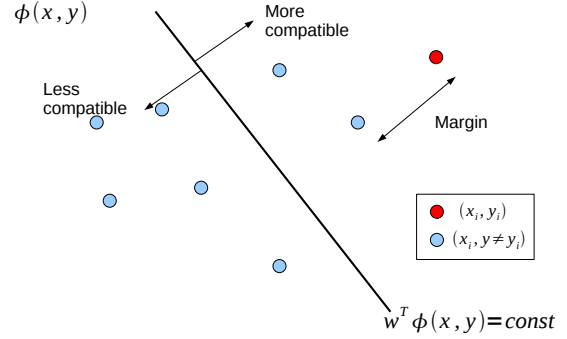
In this section we present the *GOstruct* method and kernel methods for structured output spaces. But first, we provide a brief overview of binary (two-class) classification using kernel methods. A standard approach in training predictors for binary classification problems is to learn a discriminant 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 be expressed using the kernel function as  $\sum \alpha_i K(\mathbf{x}_i, \mathbf{x})$ . See a tutorial by Ben-Hur, *et. al.*<sup>4</sup> 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  $(y_1, y_2, \dots, y_k)$ , 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:

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

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  $\phi$ . Whereas in two-class classification problems the mapping  $\phi$  depends only on the input, in

the structured-output setting it is a joint function of inputs and outputs.



**Fig. 2.** The geometric view of structured-output classification. The figure represents data points  $(\mathbf{x}, \mathbf{y})$  in the joint space of inputs and outputs. We consider a given input  $\mathbf{x}_i$  and plot it in combination with different labels  $\mathbf{y}$ . This figure represents the ideal case: The correct label has the highest compatibility value 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  $\phi(\mathbf{x}, \mathbf{y})$  (the line  $\mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) = const$  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 reparameterization of the compatibility function in terms of the coefficients  $\alpha$ :

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

where  $K : (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \rightarrow \mathbb{R}$  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:

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

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 our experiments, we have 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 is all possible combinations of functions defined by a few thousand GO terms. Explicitly enumerating all of them is not practical due to their exponential number. 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. We further reduce this number in several ways.

During training we limited this space 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).

In the case where we have homology information from BLAST hits we consider two additional output spaces for inference of test example labels,  $\mathcal{Y}_2$  and  $\mathcal{Y}_3$ , in order to examine the effect of the size of the search space on prediction accuracy. 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 leaf nodes represented in  $\mathcal{Y}_3(\mathbf{x})$  and considering all macro-labels consisting of three leaf nodes at the most. 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<sup>7</sup>. We introduce a variant of the algorithm that incorporates the concept of a margin, and pro-

pose to incorporate the loss function during 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  $\gamma$ :

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

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 2.1. The *GOstruct* method that uses it is referred to as *GOstruct<sub>p</sub> $\Delta$* , whereas the *GOstruct* method that uses the standard perceptron update is called *GOstruct<sub>p</sub>*. 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  $\alpha$  corresponding to the misclassification is decremented by 1 regardless of whether the classifier made a big mistake or a slight one<sup>7, 25</sup>. Intuitively, we would like to penalize gross misclassifications with larger values than slight errors. We propose to update the  $\alpha$  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  $\alpha$  coefficient will be updated by -1, as per 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

---

**Algorithm 2.1** 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  $i = 1, \dots, n$  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

---

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<sup>31</sup>:

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

Unfortunately, this generally results in over-constrained problems with no solutions. To get around this, we employ the *n-slack formulation* of the problem<sup>31</sup>, where we allow for some amount of margin violation. The amount of violation is represented by the slack variables  $\xi_i$ , which we add to the minimization criterion:

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

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 slack rescaling, the slack variables are scaled by the loss function, effectively relaxing the constraints for closely related outputs. The corresponding  $GOstruct$  method is referred to as  $GOstruct_{nsvm}$ . In our experiments we have also considered the margin rescaling formulation<sup>31</sup>, which

achieved results comparable to those that employed slack rescaling.

We have also implemented the *1-slack cutting-plane formulation* of the problem with both margin and slack re-scaling. This formulation is the latest development in structured SVMs<sup>13</sup> and has the advantage over the original *n-slack formulation* in that the number of constraints—and, thus, the number of the associated dual parameters—does not depend on the size of the dataset, because there is no longer a slack variable associated with each data example. The cutting-plane formulation directly enforces the observation that the average margin violation will not exceed the average slack<sup>13</sup>. The results are reported for slack re-scaling only (a formulation we refer to as  $GOstruct_{1svm}$ ) with the margin re-scaling formulation yielding comparable performance.

To train a structured SVMs, we used the working set approach<sup>31</sup> with an SMO-like algorithm<sup>23</sup> as the underlying optimizer. Like the perceptron algorithm, all structured SVMs formulations were expressed in terms of the dual coefficients  $\alpha$  instead of a vector  $\mathbf{w}$ . We refer the reader to the original papers<sup>31, 13</sup> for details.

### 3. DATA PREPARATION AND EXPERIMENTAL SETUP

We performed two experiments: one to compare the structured-output methods to homology-based transfer-of-annotation, and another to compare its

performance on the Mousefunc dataset.

### 3.1. Four species prediction-by-sequence-similarity experiment

To compare the structured-output 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<sup>34</sup>, 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. In developing our methods we used the GOslims ontology; we report results on the full GO ontology, thereby avoiding overfitting our test data. In our analysis we considered all GO molecular function terms that appear as annotations in at least 10 proteins, resulting in a total of 361 nodes.

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<sup>26</sup>. This was done by removing all annotations with the evidence codes: IEA, ISS, ND, RCA, and NR. Note that limiting the experiments to annotations that were possibly derived by computational means limits the number of species that can be considered to a very small set of model organisms, and for simplicity we focused on the eukaryotes listed above.

We then ran BLAST for each of the proteins in our dataset against all four species, removing the hits where the 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 structured-output 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 empirical kernel map<sup>28</sup>; more specifically, each feature was the negative-log of the BLAST  $e$ -values that are below 50, where the features were 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  $\gamma$  (for perceptron) and  $C$  (for structured SVM) for each left-out species. In our experiments, we noticed that finding the right value of  $\gamma$  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<sup>22</sup>. In particular, we used 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).

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

**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 two SVM variants ( $GOstruct_{svmm}$  and  $GOstruct_{svm}$ ) 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.

Test on	<i>C. elegans</i>	<i>D. melanogaster</i>	<i>S. cerevisiae</i>	<i>S. pombe</i>	Output Space
# proteins	844	1804	1853	898	
BLAST-NN	0.628	0.450	0.381	0.360	
$GOstruct_p$	0.634	0.446	0.414	0.426	$\mathcal{Y}_1$
$GOstruct_p$	0.627	0.438	0.391	0.381	$\mathcal{Y}_2$
$GOstruct_p$	0.628	0.436	0.388	0.371	$\mathcal{Y}_3$
$GOstruct_{p\Delta}$	0.589	0.378	0.389	0.372	$\mathcal{Y}_1$
$GOstruct_{p\Delta}$	0.605	0.368	0.358	0.341	$\mathcal{Y}_2$
$GOstruct_{p\Delta}$	0.623	0.415	0.368	0.339	$\mathcal{Y}_3$
$GOstruct_{1svm}$	0.628	0.420	0.390	0.368	$\mathcal{Y}_1$
$GOstruct_{1svm}$	0.634	0.416	0.366	0.341	$\mathcal{Y}_2$
$GOstruct_{1svm}$	0.640	0.419	0.366	0.347	$\mathcal{Y}_3$
$GOstruct_{nsvm}$	0.625	0.433	0.382	0.366	$\mathcal{Y}_1$
$GOstruct_{nsvm}$	0.634	0.425	0.364	0.338	$\mathcal{Y}_2$
$GOstruct_{nsvm}$	0.637	0.424	0.365	0.345	$\mathcal{Y}_3$
Random	0.821	0.868	0.877	0.816	

other.

The values of  $\gamma$  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 ( $GOstruct_p$ ). 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 perfor-

mance 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.

The two SVM formulations performed equally well, with the 1-slack formulation performing slightly better on *D. Melanogaster*. Both methods outperform BLAST-NN and  $GOstruct_p$ , but not  $GOstruct_{p\Delta}$ , which is a significantly simpler algorithm. While only the slack re-scaling results are reported, similar performance was achieved with margin re-scaling.

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 BLAST-NN and the random classifier were 0.004 and 0.009, respectively.  $GOstruct_p$  had standard deviation values in the range (0.006, 0.01) for the different output spaces;  $GOstruct_{p\Delta}$  yielded more consistent performance with the standard deviation values in the range (0.003, 0.007).

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.

## 4.2. Mousefunc experiment

The results on the Mousefunc dataset are presented in Table 3; the number of test examples and number of GO terms are given in Table 2. We applied  $GOstruct_{p\Delta}$  and the two versions of structured SVMs for prediction of GO terms in each of the three namespaces and compared our results to the predictions submitted by the participants in the Mousefunc challenge. First we consider classifier error as measured by the average loss per sample. Under this measure the  $n$ -slack formulation of the structured SVM consistently outperforms all other algorithms. Three of the top performing algorithms (Alg2, Alg3, and  $GOstruct$ ) used kernel methods, integrating results across the hierarchies in different ways. In addition to the comparison with the Mousefunc challenge results we compare the  $GOstruct$  method with the naive SVM approach of training separate binary classifiers for each node in the GO hierarchy. This experiment was performed using the SVM implementation in the PyML machine learning library available at `pyml.sf.net` run with the default parameters, and the same input-space kernel used to assess the  $GOstruct$  methods. The SVM-based  $GOstruct$  method outperforms the collection of binary SVMs when performance is measured by the kernel loss. These results are not surprising: the  $GOstruct$  method is trained to optimize the kernel loss, whereas the other methods optimize other criteria.

We also report the average area under the ROC curve (AUROC) for all experiments<sup>24</sup>. For  $GOstruct$  algorithms, the confidence measure used in AUROC computations was taken to be the difference in the compatibility function values between the predicted label and a label that had the corresponding node

prediction flipped. The values are reported in the bottom half of Table 3. The geneMANIA method yielded the best AUROC, which is consistent with the literature report<sup>22</sup>. It is also the algorithm that yields the worst average loss per sample. This suggests that training a classifier to predict individual node annotations (as measured by the AUROC) does not solve the more general problem of predicting the full annotation (as measured by the kernel loss). The claim is further strengthened by the similar trend observed in  $GOstruct_{nsvm}$  versus the binary-SVM single-node approach. The structured-output algorithm yields better loss values but worse AUROC. We were surprised to find, however, that the naive two-class SVM performed comparably or better than all the Mousefunc methods employing a probabilistic algorithms to combine the single-node predictions (as done in Alg 2, Alg 3, and Alg 6).

**Table 2.** Statistics of the Mousefunc dataset.

GO namespace	number of terms	test example
MF	205	531
BP	513	626
CC	119	307

The  $GOstruct_{p\Delta}$  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. In our experience, the perceptron converged in as few as five passes through the training data. On a modern machine, this took roughly three hours with pre-cached kernel matrices. While being significantly simpler than all other algorithms, the structured perceptron maintained competitive performance with all the other entries and consistently outperformed two of the submitted algorithms.

We note that all of the algorithms performed best when tasked with the prediction of molecular function, followed by cellular component, with worst performance on prediction of the biological namespace terms. Biological process is the namespace that had the largest number of terms, which explains in part the worst performance in predicting it—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 cellu-

**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 (top), and the average area under the ROC (AUROC) curve per node (bottom) for each algorithm. Lower values of the loss and higher values of the AUROC are better. The best value for each experiment is highlighted. Alg 1 denotes the work by Kim, *et al.*<sup>14</sup>. Alg 2 is an ensemble of calibrated SVMs by Obozinski, *et. al.*<sup>20</sup>. Alg 3 is the kernel logistic regression, submitted by Lee, *et al.*<sup>17</sup>. Alg 4 is geneMANIA<sup>19</sup>. Alg 5 is GeneFAS<sup>6</sup>. Alg 6 is the work by Guan, *et al.*<sup>11</sup>.  $GOstruct_{p\Delta}$  uses the perceptron algorithm (Algorithm 2.1), and  $GOstruct_{1svm}$  and  $GOstruct_{nsvm}$  denote the 1-slack and  $n$ -slack formulation of the structured SVMs with slack 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.

Performance measure	GO namespace	Literature						$GOstruct$			SVM (single node)
		Alg 1	Alg 2	Alg 3	Alg 4	Alg 5	Alg 6	$p\Delta$	$1svm$	$nsvm$	
kernel loss	MF	0.633	0.470	0.431	0.668	0.522	0.618	0.526	0.652	<b>0.362</b>	0.417
	BP	0.788	0.687	0.628	0.839	0.735	0.717	0.680	0.797	<b>0.594</b>	0.671
	CC	0.599	0.602	0.534	0.755	0.666	0.664	0.617	0.670	<b>0.490</b>	0.587
AUROC	MF	0.861	0.872	0.888	<b>0.916</b>	0.859	0.904	0.803	0.747	0.880	0.915
	BP	0.761	0.737	0.770	0.834	0.755	<b>0.839</b>	0.705	0.628	0.730	0.787
	CC	0.790	0.767	0.778	<b>0.842</b>	0.750	0.834	0.754	0.701	0.807	0.815

lar component and biological process namespace was very similar, and as we observe here, performance was much better for predicting molecular function<sup>26</sup>.

We were surprised by the relatively poor performance of the 1-slack SVM. The version that employs slack re-scaling managed to stay on par with the lower end of the submitted algorithms, while the margin re-scaling version (results not shown), was outperformed by every other algorithm. The 1-slack SVM has a single  $\alpha$  coefficient associated with a collection of inputs/labels, as opposed to the  $n$ -slack formulation which has a coefficient associated with individual inputs/label pairs. We believe this added flexibility in the  $n$ -slack formulation was the factor that helped its performance.

## 5. CONCLUSIONS

In this paper we presented the  $GOstruct$  method for predicting GO terms by explicitly modeling the structure of the GO hierarchy using kernel methods for structured output spaces, a novel development in the field of machine learning. A very simple version of the  $GOstruct$  method which uses the perceptron algorithm performs better than a transfer-of-annotation method when provided the same information. Additionally, we demonstrated improved performance on the Mousefunc benchmark using a more elaborate version of the  $GOstruct$  method that employed structured SVMs. The perceptron-based version also yielded competitive performance to some of the more elaborate methods submitted by the participants.

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. We also proposed a generalization of the  $F_1$  loss function<sup>31</sup> to arbitrary output spaces through the use of kernels, and a variant of the perceptron update rule that leverages the loss function to assess the necessary amount of update. We demonstrate empirically that the modified rule leads to improved 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. Tsochantaris, and T. Hofmann. Hidden markov support vector machines. *Proc. ICML*, 6, 2003.
3. Z. Barutcuoglu, R.E. Schapire, and O.G. Troyanskaya. Hierarchical multi-label prediction of gene function. *Bioinformatics*, 22(7):830–836, 2006.
4. A. Ben-Hur, C.S. Ong, S. Sonnenburg, B. Schölkopf, and G. Rätsch. Support Vector Machines and Ker-

- nels for Computational Biology. *PLoS Computational Biology*, 4(10), 2008.
5. N. Cesa-Bianchi, C. Gentile, and L. Zaniboni. Incremental algorithms for hierarchical classification. *The Journal of Machine Learning Research*, 7:31–54, 2006.
  6. 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.
  7. 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.
  8. M. Deng, T. Chen, and F. Sun. An integrated probabilistic model for functional prediction of proteins. In *RECOMB*, pages 95–103, 2003.
  9. 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.
  10. Gene Ontology Consortium. Gene ontology: tool for the unification of biology. *Nat. Genet.*, 25(1):25–9, 2000.
  11. 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.
  12. T. Hofmann, L. Cai, and M. Ciaramita. Learning with taxonomies: Classifying documents and words. *NIPS Workshop on Syntax, Semantics, and Statistics*, 2003.
  13. T. Joachims, T. Finley, and Chun-Nam Yu. Cutting-plane training of structural svms. *Machine Learning*, to appear.
  14. 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.
  15. 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.
  16. 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.
  17. 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.
  18. 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.
  19. 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.
  20. 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.
  21. D. Pal and D. Eisenberg. Inference of protein function from protein structure. *Structure*, 13:121–130, January 2005.
  22. 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.
  23. J. Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. *Advances in Kernel Methods-Support Vector Learning*, 208, 1999.
  24. Foster J. Provost, Tom Fawcett, and Ron Kohavi. The case against accuracy estimation for comparing induction algorithms. In J. Shavlik, editor, *ICML '98: Proceedings of the Fifteenth International Conference on Machine Learning*, pages 445–453, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.
  25. V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and inference over constrained output. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1124–1129, 2005.
  26. M. Rogers and A. Ben-Hur. The use of gene ontology evidence codes in preventing classifier assessment bias. *Bioinformatics*, 25(9):1173–1177, 2009.
  27. J. Rousu, C. Saunders, S. Szedmak, and J. Shawe-Taylor. Kernel-based learning of hierarchical multi-label classification models. *The Journal of Machine Learning Research*, 7:1601–1626, 2006.
  28. B. Schölkopf, J. Weston, E. Eskin, C. Leslie, and W.S. Noble. A kernel approach for learning from almost orthogonal patterns. *Proceedings of the 13th European Conference on Machine Learning*, pages 511–528, 2002.
  29. 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.
  30. B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. *Advances in Neural Information Processing Systems*, 16:51, 2004.

31. 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.
32. K. Tsuda, H.J. Shin, and B. Schölkopf. Fast protein classification with multiple networks. In *ECCB*, 2005.
33. CJ Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann Newton, MA, USA, 1979.
34. A. Vinayagam, R. König, 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.